INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

Bell & Howell Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA

UMI®
800-521-0600

OPERATING SYSTEM SIMULATION (OSS) IN JAVA: THE SYSTEM ARCHITECTURE

MING HU

A MAJOR REPORT
IN
THE DEPARTMENT
OF
COMPUTER SCIENCE

PRESENTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF COMPUTER SCIENCE
CONCORDIA UNIVERSITY
MONTREAL, QUEBEC, CANADA

AUGUST 1998

Abstract

Operating System Simulation (OSS) in Java: The System Architecture

Ming Hu

The OSS system is an operating system simulation tool. The simulated operating system is a multi-process system running on a single processor. The underlying simulated hardware contains CPU, registers, main memory, backing-store and other system resources controlled by semaphores. The simulated operating system itself contains six major components, which are process management, scheduler, memory management, message management, semaphore management and resource management. The OSS system can be used to help computer science students to get a clear understanding of how an operating system works and to study the impact of various algorithms on the performance of an operating system. As the implementing language, Java has outstanding features, such as the multiple threads programming, garbage collection, abstract window toolkit (AWT) and the portability, which are very much appreciated in the development of the OSS system.

Acknowledgments

I would like to express my deepest gratitude to my supervisor, Dr. Peter Grogono. His guidance and encouragement made this project work a pleasant and extremely educational experience.

Table of Contents

List of Figures

1. Introduction

An operating system is a program that acts as an intermediary between a user of a computer and the computer hardware. It provides an environment in which a user can execute programs and it is an essential part of a computer system. Similarly, a course on operating systems is an essential part of a computer science education. An intuitive and interesting learning tool will certainly help students to get a clear understanding of how an operating system works.

The main goals of an operating system are to make a computer system convenient to use and to have the computer hardware used in an efficient manner. With a flexible simulation environment, analyzing the impact of various algorithms on the performance of an operating system becomes an easy task.

Nowadays, almost all operating systems are written in a systems-implementation language or in a higher-level language. This feature has improved the implementation, the maintenance and the portability of an operating system.

Having chosen Java as the implementing language of the OSS system, we are impressed by the capability of Java in the development of a computer software system with its advanced features, such as the multiple threads programming, the garbage collection, the abstract window toolkit (AWT) and the portability.

The rest of this report is organized as follows:

- Section 2, "**Operating System Simulation**", describes the rationale of the operating system simulation.
- Section 3, "**Related Work**", gives a brief account of another OS simulation called ToyOS.
- Section 4, "**Implementation Language - Java**", describes the distinct features of Java.
- Section 5, "**System Architecture**", describes the design and implementation of OSS system.
- Section 6, "**Conclusion**", summarizes the key features of the OSS system and proposes the future enhancements.
- "**References**", gives the references used in the development of the OSS system.
- "**Appendix**", gives a complete list of OSS system source codes.
- Other aspects of OSS system are described in the report "Operating System Simulation (OSS) in Java: The User Interface" by Hong Wei Mao [8].

2. Operating System Simulation

In general, computer simulation is the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output. Simulation embodies the principle of "learning by doing" - to learn about the system we must first build a model and make it run.

Computer simulation is a highly interdisciplinary field since it is widely used in all aspects of industry, government and academia. One can find the teaching of simulation techniques in almost every academic department from economics and social science to engineering and computer science.

Simulations have two key advantages over actual programs. The first is that actual programs tend to keep marching on by. Simulations allow students, researchers and others to play with time in ways the actual programs do not permit. Often, the actual programs move so quickly that users do not have time to think things over as much as they would like. However, in a simulation, if users wish to sit and ponder their courses of action, they can freeze the simulation, and perhaps even ask an expert some questions. If users are unclear as to why things turned out the way they did, we can allow them to loop back in time and review the course of events. If events are moving too quickly, users can slow them down. Users can even decide to back time up so that can try a different approach.

Due to the important role that an operating system performs, the study of operating systems has traditionally been one of the most challenging and exciting software disciplines in computer science. The largest challenges in an operating systems class are to make complex aspects of operating systems easy to understand and avoid making abstract aspects boring. The goal of implementing an operating system simulation is to meet this challenge.

A good operating system simulation can help students and researchers to develop an understanding and appreciation for the fundamentals behind most operating systems. Students can find the best way to learn considerable amounts about operating system by experimenting on their own through an operating system simulation. There are several ways in which an OS simulation can help its users. First, an intuitive and interesting simulation tool will certainly help students to get a clear understanding of how an operating system works. Second, with a flexible simulation environment, analyzing the impact of various algorithms on the performance of an operating system becomes an easy task. Third, through the object-oriented design environment, it would be easy to add or change some features for specific systems to get a better simulation effect.

3. Related Work

A ToyOS developed in Modula-3 was used in the operating systems course (COMP 346/546, Summer 1997) in the Department of Computer Science at Concordia University. The ToyOS simulates a simple multi-programming environment and includes process management, CPU scheduling and memory management. A complete list of ToyOS system source codes can be found in Hong Wei Mao's report [8].

The ToyOS contains the following major components:

- Module User. This module contains some test procedures, which invokes system calls and the simulation.
- Module SysCall. This module implements some system calls such as Fork, Getpid, and Yield. The system calls use registers to pass values to the operating system and to get the returned values.
- Module OS. This module provides the entry points of the operating system layer.
- Module MyProcess. This module is responsible for process creation, execution, switching and termination.
- Module MyScheduler. This module manages the ready-to-run processes and decides who gets to run next, whenever a scheduling decision has to be made.
- Module MyMemMgr. This module is responsible for allocating a given number of frames to the process and freeing the frames allocated to a process.
- Module MachineOS. This module provides a shortcut to the hardware.

- Module Machine. This module provides the hardware emulation. It includes the following emulation: registers, main memory, backing store, DMA emulation and clock.

We argue that the OSS system is better than the ToyOS system for the following reasons. First, more operating system components and algorithms are implemented in the OSS system. Second, a graphical user interface is implemented in the OSS system. Users can observe some internal operating system data structures through the user interface. Third, a configuration manager is implemented in the OSS system, and this provides users with great convenience of shaping the OSS system in the way they want. Fourth, Java is chosen to be the implementing language for the OSS system. The advantages of using Java are described in detail in the following section.

4. Implementing Language - Java

Java is chosen to be the implementing language. In general, a good simulation implementing language should at least hold the following aspects:

- Providing a mechanism to handle user's interaction with the system and internal system computation simultaneously.
- Providing a development toolkit, which facilitates building application user interface.
- Containing object oriented technologies that simplify software development and maintenance.
- Having a high portability that enables an application to run on different platforms.

Java is a new programming language, with elements from C, C++ and other languages, and with libraries highly tuned for the Internet environment. It draws inspiration from many sources such as Mesa, Modula-3, Lisp, Objective C, C and so on. The overall ability of Java makes it well qualified for implementing a simulation. Some distinct features of Java are discussed in the following six paragraphs.

4.1. Multiple Threads

Threads simply extend the concept from switching between several different programs to switching between several different functions executing simultaneously within a single program. Instead of the costly overhead of saving the state (virtual memory map, file

descriptors, etc.) of an entire process, a low-overhead context switch (saving just a few registers, a stack pointer, the program counter, etc.) within the same address space is done [4].

The Java development environment supports multithreaded programs through the language, the libraries, and the runtime system. Java has a lot of features for threads such as the life cycle of a thread, scheduling threads, grouping threads, and synchronizing threads.

4.2. Garbage Collection

Garbage collection means the automatic reclaiming of memory that is no longer in use. In C language, *malloc()* allocates memory, and *free()* makes it available for reuse. In C++, the *new* and *delete* operators have the same effect. Both of these languages require explicit de-allocation of memory. The programmer has to tell what memory to give back, and when. In practice, this has turned out to be an error-prone task.

Java takes a different approach. Instead of requiring the programmer to take the initiative in freeing memory, the job is given to a runtime component called the garbage collector. It is the job of the garbage collector to sit on top of the memory heap and periodically scan it to determine objects that are not being used any more. It can reclaim that memory and put it back in the free store pool within the program [4].

Taking away the task of memory management from the programmer gives him or her one less thing to worry about, and makes the resulting software more reliable in use. It may take a little longer to run Java programs as compared to programs written in a language like C++ with explicit memory management. On the other hand, it is much quicker to debug and get the program running in the first place.

4.3. The Abstract Window Toolkit (AWT)

The Java AWT provides a set of primitive objects for partitioning the display area and for simple interactive objects such as buttons, text fields, lists etc. We can thus build a user interface quickly.

The AWT is a set of classes and packages, which can be used without concern for platform specific issues. The AWT leaves the actual rendering and behavior of different components to the native windowing systems.

The platform dependent work is done by the *java.awt.peer* classes. An AWT peer is just a wrapper around native code (platform dependent code). This allows application windows to have the same look and feel as the platform windowing environment.

4.4. Applet

Java is a fine general purpose programming language. It can be used to good effect to generate stand-alone executables, just as C, C++, Visual Basic, Pascal or Fortran can. Java offers the additional capability of writing code that can live in a web page and then get downloaded and executed when the web page is browsed [4].

The OSS system is implemented as a stand-alone application and can be transferred to an applet with little difficulty.

4.5. "Almost" Platform Independence

The "write once, run anywhere" slogan is synonymous with the Java programming language. The Java run-time library provides application developers with the ability to write code in one single language and the confidence that the code will execute without modification on virtually any hardware, any operating system, and any application environment. Developers are freed from the costs of porting their applications to a myriad of target platforms and worse, of maintaining those multi-code bases. Powerful cross-platform run-time systems have existed for many other interpreted languages. However, Java is the first to achieve widespread popularity among commercial developers [6].

In general, Java's accomplishments toward platform independence are rather amazing, but not quite perfect. As a shining example, when Sun Microsystems announced the 1.0 release of its Java development environment "Java Workshop", it was touted as "completely written in Java". It turns out that Java Workshop now only runs on Windows and Solaris platforms (not exactly platform independent) [7].

The worst thing about architecture independence is that it keeps you independent of the architecture - meaning that Java needs to assume the lowest common denominator of available resources. Java supports only one mouse button because Macs only have one mouse button. Java can only assume very generic features of the host system: it assumes a system has a CPU, memory, and a graphics subsystem. It also assumes the operating system is multithreaded, which negates the idea of Java on Windows 3.1. Joysticks, special keypads, and the like can not be accessed from Java [7].

To get system-specific, you need to write some C code and interface it with Java. The problem is that if your program is even 1% C code, you lose Java's platform independence and its solid security model [7].

4.6. Slow Performance

Java is slow. The reason for Java's speed can be simplified in one word: abstraction. To OO/high-level language designers, abstraction is the goal. To the computer, any abstraction puts the code another step away from what it understands. It is also the

computer's job to decode those steps down to its machine language so it can actually run the program [7].

Machine language coding is the lowest level we can code in. Above that, we can abstract machine code to assembly code. Then the computer geniuses invented the high-level language abstraction. Java puts the object-orientation abstraction on top of this. Finally, Java adds the platform-independence abstraction. That's a lot of abstraction and it takes a lot of compilers, optimizers, and smart run-time environments to remove them all, to get our program back down to machine code so it can actually run.

How could Java's performance be improved? The compiler companies came out with their just-in-time (JIT) compilers. These programs (which are closer to assemblers than compilers) convert Java's stack-based intermediate representation into native machine code immediately prior to execution on your machine. That way, the Java program actually runs as a real executable. JIT compilers do speed up Java programs. Java performance could someday approach C or C++ performance [7].

5. System Architecture

5.1. Design

From the system design point of view, the OSS system consists of four components. They are user interface, operating system, system configuration and bookkeeper, as shown in Figure 1.

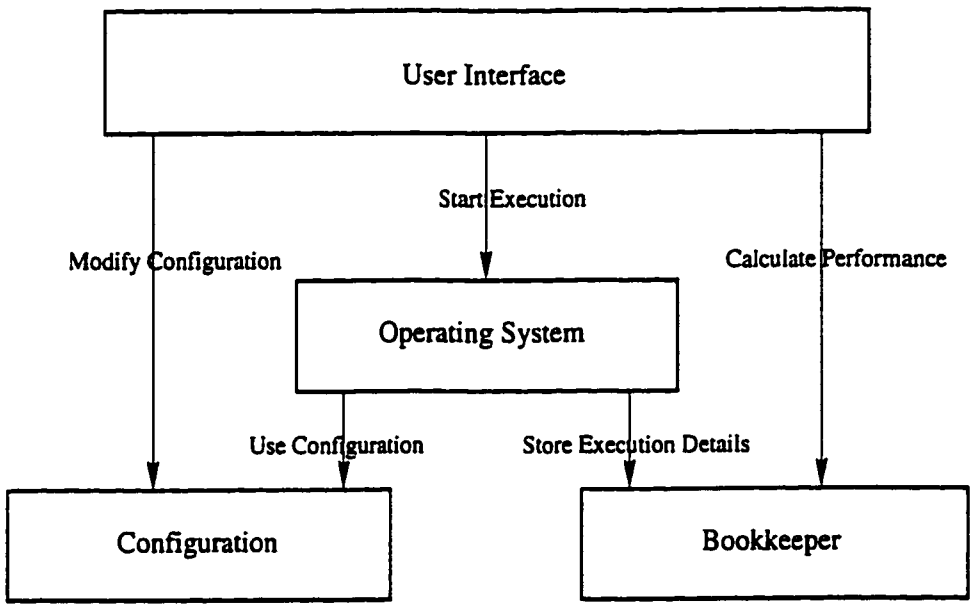

Figure 1. System Topology

The user interface is actually the control center of the OSS system. It has two major functions. The first functionality is to handle the user's interactions with the OSS system. This includes changing system configuration, starting a round of execution of operating system, interfering in the execution of operating system, obtaining help from the OSS

system and shutting down the OSS system. The second functionality is to monitor some important internal data structures and system performance of the operating system. These include current running process, ready queue contents, sleep queue contents, message waiting queue contents, semaphore waiting queue contents, CPU utilization, throughput, response time, turnaround time, waiting time, page fault rate, memory availability and operating system execution history.

The simulated operating system is the core of the OSS system. It is a multi-process system multiplexed on a single processor and it runs on top of the simulated hardware. The simulated hardware contains CPU, registers, main memory, back store and the system resources controlled by the semaphores. The operating system itself contains six major components. They are process management, scheduler, memory management, message management, semaphore management and resource management.

The system configuration consists of system parameters, on which the execution of whole OSS system is based. These system parameters are divided into six catalogs. The first catalog contains OSS system related parameters, such as maximum number of history items reflecting the execution details of the simulated operating system and clock ticks of DMA delay. The second catalog contains memory management related parameters, such as system memory capacity and page replacement algorithm. The third catalog contains process management related parameters, such as maximum number of processes allowed in the operating system and scheduling algorithm. The fourth catalog contains inter process communication related parameters, such as number of semaphores and number of

keys for message passing among processes. The fifth catalog contains the system parameters regarding the simulation mode. The sixth and last catalog contains system parameters regarding the simulation results output.

The bookkeeper records the data needed to calculate the operating system performance. These data include the trace of access to CPU, every page fault error, the exit time of a process, the execution time (from submission to exit) of a process, the response time (from submission to first execution) of a process and the time a process spent in the sleep queue.

As shown in Figure 2, there are three threads defined in the OSS system. They are OSS thread, Simulation thread and Interface thread. OSS thread is responsible for drawing the user interface and handling the user's interactions with the OSS system. Simulation thread is responsible for the execution of operating system. Interface thread is responsible for updating those parts of user interface that monitor the operating system performance.

When the OSS system is initially started, OSS thread is the only thread running in the system. It draws the user interface and waits for the user's commands. At this time, the user can change system configuration, on which the execution of operating system is based. When the start simulation command is issued (a click on "Start" button), the Simulation thread is spawned to set up operating system component classes in a specific order and start the execution of the operating system. The behavior of the operating system is determined by the system configuration, which was characterized by the user.

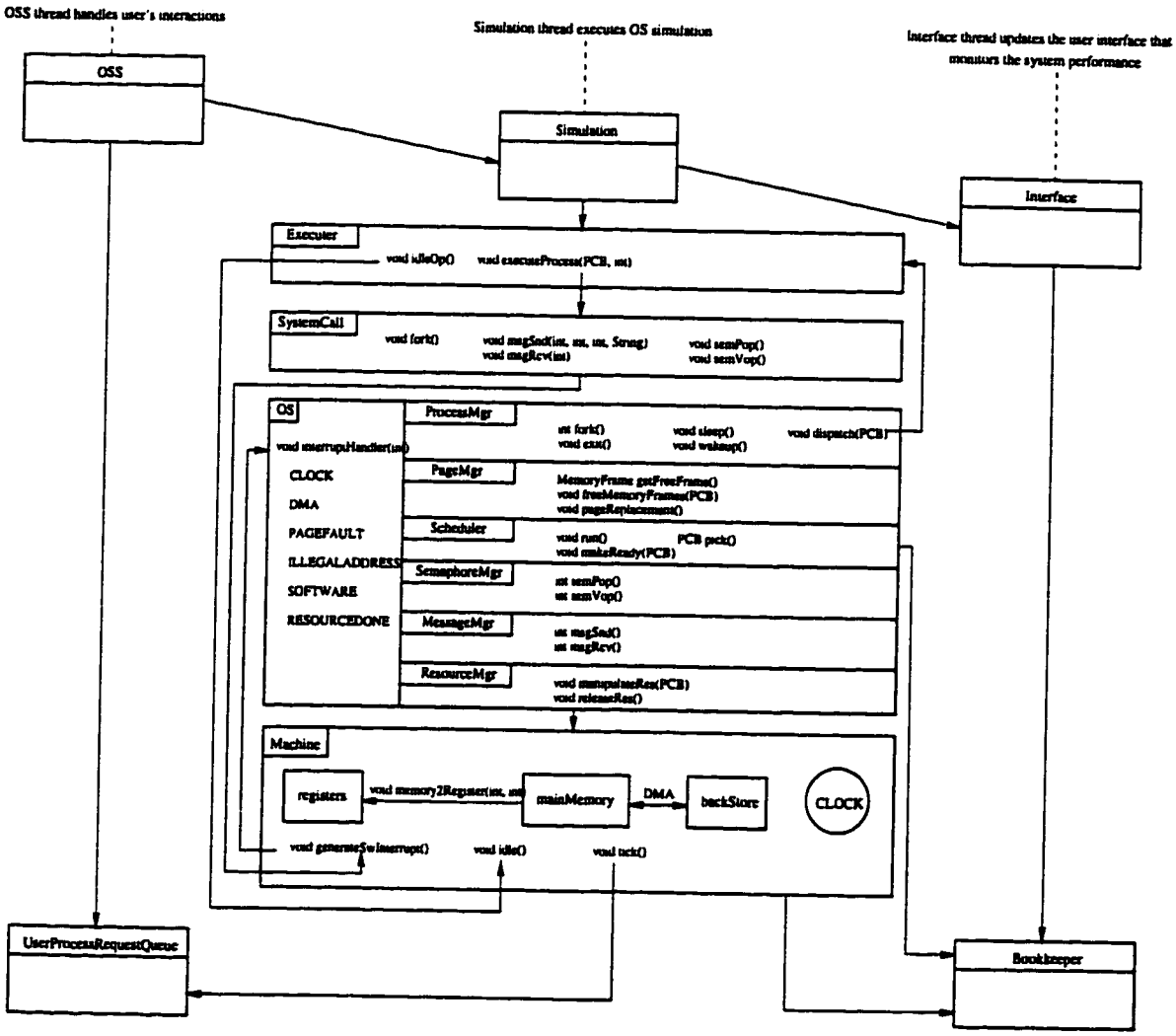

Figure 2. System Architecture

The execution details of the operating system are kept in the bookkeeper. The Interface thread, created by the Simulation thread, uses the data kept in the bookkeeper to calculate the operating system performance and update the user interface accordingly.

5.2. Implementation

5.2.1. System clock

A simulated system clock interrupts the operating system execution periodically and gives everything a chance to happen. Time is advanced by one unit each time the *tick()* method of *Machine.java* class is called. Each round of execution of a process (either a user process or the idle process) is followed by the call for *tick()*.

What *tick()* does is summarized as follows.

- Updating the DMA counter. This happens only when there is a DMA operation pending. DMA operation is considered finished when a certain amount of time has passed.
- Updating the semaphore counters. Each counter corresponds to a resource occupied by a user process. The occupation of a resource is considered finished when a certain amount of time has passed.
- Generating a clock interrupt. This is to give the operating system a chance to decide what to do next periodically.

- Generating a DMA interrupt. This happens when the DMA counter expires.
- Generating a resource done interrupt. This happens when any of the semaphore counters expires.
- Checking the user process request queue. For each request in the queue, a software interrupt is issued to create a new user process. This is the best place to check the request queue from the Simulation thread to keep the OSS system responsive to the user's request to add a new process.

5.2.2. User process

UserProcessRequest.java class models the user process requests. There are four key pieces of information stored in this class. They are: whether or not there is a memory operation required, whether or not there is a message operation required, whether or not there is a semaphore operation required, and the priority level if the priority algorithm is chosen as the scheduling algorithm. The object of *UserProcessRequest.java* can be specified by the user through *UserProcessRequestDialog.java* class in the interactive execution mode and put into *UserProcessRequestQueue.java*. At the same time, the object of *UserProcessRequestQueue.java* is checked periodically by the Simulation thread, precisely, the *tick()* method of *Machine.java* class. For every request in the queue, *tick()* issues a software interrupt which forks a new user process and puts it in the ready queue. An object of *UserProcess.java* class, whose attributes encapsulate memory, message, semaphore operations and the order of these operations, represents the newly created user process. For example, a vector, which contains randomly generated integers,

represents the memory addresses if a memory operation is specified. When a memory access is done, the corresponding memory address is removed from the vector. The information encapsulated in the *UserProcess.java* class is understood by *Executer.java* class and the object of *Executer.java* class is called every time a process is put into the running state.

5.2.3. Different page managers and schedulers

PageMgr.java and *Scheduler.java* classes model the two major components, the page manager and the scheduler, of the operating system. They are defined as abstract classes, which are the super classes of those algorithm-dependent page managers and schedulers. *PageMgrFIFO.java* and *PageMgrSC.java* classes, which model page managers implementing a FIFO page replacement algorithm and a second chance page replacement algorithm respectively, are the only subclasses of *PageMgr.java* class in the current implementation. *SchedulerPR.java* and *SchedulerRR.java* classes, which model schedulers implementing round robin algorithm and priority algorithm, are the only subclasses of *Scheduler.java* class in the current implementation. The abstract classes are so general that you can think of them more as a framework for other classes than as classes with specific instances you want to use. The use of abstract classes makes the design of classes cleaner. More algorithm-dependent page managers and schedulers could be added into the system with minimal effort under current design in the future.

5.2.4. System configurations

The OSS system provides users with maximum convenience of shaping the system in the way they want. *ConfigCurrent.java* class keeps the configuration information used by the system. *ConfigDefault.java* class keeps the default configuration information in case that users feel getting lost in shaping the system and want to go back to play with the best tuned OSS system. *Configuration.java* class provides means for users to change the configuration information. Due to the complexity of the OSS system itself, it is not surprising that there are quite a number of configuration items that need to be set up. How to present these items inside a restricted window area in a professional way is a challenging task for programmers of some programming languages. However, this is not the case for Java programmers. Thanks to Java's CardLayout manager, an elegant looking "tabbed dialog" window is implemented in the OSS system. The same window area is used to display six different categories of configuration information. The switching of six categories is easily done by clicking of corresponding buttons.

5.2.5. A separate bookkeeper class

As its name indicates, this class acts as a bookkeeper to keep operating system execution details that are needed to calculate the system performance. The Simulation thread uses this class to store operation system execution details. The Interface thread uses this class to calculate the system performance. The simultaneous access to bookkeeper class by

Simulation thread and Interface thread may cause the corruption of data stored in the bookkeeper class. This kind of consistency problem is easily solved using Java's synchronization feature. You just have to define those methods, which may cause the problem, as synchronized methods. Java will do the rest for you.

6. Conclusion

6.1. Summary

The OSS system consists of two major components:

- The user interface;
- The underlying simulated operating system.

The underlying simulated operating system is a multi-process system running on a single processor and it runs on top of the simulated hardware.

The simulated hardware contains CPU, registers, main memory, backing-store and the system resources controlled by the semaphores.

The simulated operating system itself contains six major components. They are: process management, scheduler, memory management, message management, semaphore management and resource management.

The current implementation of the OSS system has following features:

- Two execution modes: interactive and non-interactive.
- Two page replacement algorithms: FIFO and second chance.
- Two scheduling algorithms: round robin and priority.
- Flexibility of specifying system configurations.
- Interfering in the system execution speed.

- Providing help.
- Monitoring the simulation states.
- Monitoring the simulation performance.
- Multiple operations for a user process.
- Output of the simulation results to a file.
- Java inheritance – easy to add more operating system algorithms.
- Java portability – platform independent.

6.2. Future Work

The following enhancement in the future are proposed:

- More scheduling and page replacement algorithms.
- Animations on the representation of system internal states and performance criteria.
- Virtual terminals that can take users' commands.
- Communications among processes running on different hosts.

References

1. Abraham Silberschatz, Peter B. Galvin, "***Operating System Concepts***", Forth Edition, Addison-Wesley, 1994.
2. A ***ToyOS*** program implemented in Modula-3 and used by operating systems course (COMP 346/546, Summer 1997) in the Department of Computer Science at Concordia University.
3. Gary Cornell, Cay S. Horstmann, "***Core Java***", Second Edition, Sun Microsystems, Inc., 1997.
4. Peter van der Linden, "***Just Java***", Second Edition, SunSoft Press and Prentice Hall, 1997.
5. http://www.javasoft.com:80/products/jdk/1.1/docs/api/a-index.html
6. Sandeep Singbal, Binb Nguyen, "***The Java Factor***", Communications of the ACM, June 1998-Volume 41, Number 6.
7. Paul Tyma, "***Why Are We Using Java Again?***", Communications of the ACM, June 1998-Volume 41, Number 6.
8. Hong Wei Mao, "***Operating System Simulation (OSS) in Java: The User Interface***", Major Report, Department of Computer Science, Concordia University, August 1998.

Appendix – OSS Source Code

A.1. Bookkeeper.java

```
import java.util.*;


/* This class stores samples used to calculate system performance parameters
 * and does the actual calculations when needed. */

public class Bookkeeper{

 /* The following vectors are needed to store samples used to measure
  * the system performance. Please see Help for the definition of
  * these performance criteria. */

 private Vector CPUUtilization; // CPU utilization.
 private Vector throughput;    // Throughput.
 private Vector turnaroundTime; // Turn around time.
 private Vector responseTime;   // Response time.
 private Vector waitingTime;    // Waiting time.
 private Vector pagefaultRate;  // Page fault rate.


 /* The constructor initializes vectors. */

 public Bookkeeper(){
  CPUUtilization = new Vector();
  throughput = new Vector();
  turnaroundTime = new Vector();
  responseTime = new Vector();
  waitingTime = new Vector();
  pagefaultRate = new Vector();
 }


 /* Add samples to the CPU-utilization vector. When the number of
  * samples kept in the vector exceeds a maximum number, the oldest
  * sample is removed from the vector. */

 public synchronized void addCPUUtilization(boolean act){
  CPUUtilization.insertElementAt(new Boolean(act), 0);
  if (CPUUtilization.size() > ConfigCurrent.MAX_CPU_USE_BITS)
   CPUUtilization.removeElementAt(ConfigCurrent.MAX_CPU_USE_BITS);
 }


 /* Add samples to the page-fault-rate vector. When the number of
  * samples kept in the vector exceeds a maximum number, the oldest
  * sample is removed from the vector. */

 public synchronized void addPageFaultR(boolean act){
  pagefaultRate.insertElementAt(new Boolean(act), 0);
  if (pagefaultRate.size() > ConfigCurrent.MAX_MEM_USE_BITS)
   pagefaultRate.removeElementAt(ConfigCurrent.MAX_MEM_USE_BITS);
 }


 /* Add samples to the throughput vector. When the number of
  * samples kept in the vector exceeds a maximum number, the oldest
  * sample is removed from the vector. */

 public synchronized void addThroughput(long exitTime){
  throughput.insertElementAt(new Long(exitTime), 0);
```

```
  if (throughput.size() > ConfigCurrent.MAXPROCESS)
    throughput.removeElementAt(ConfigCurrent.MAXPROCESS);
 }


 /* Add samples to the turn-around-time vector. This is done by
  * calling a private method -- addVector. */

 public void addTurnaroundT(int execTime){
  addVector(turnaroundTime, execTime);
 }


 /* Add samples to the response-time vector. This is done by
  * calling a private method -- addVector. */

 public void addResponseT(int respTime){
  addVector(responseTime, respTime);
 }


 /* Add samples to the waiting-time vector. This is done by
  * calling a private method -- addVector. */

 public void addWaitingT(int waitTime){
  addVector(waitingTime, waitTime);
 }


 /* Calculate CPU utilization. This is done by calling a private
  * method -- computeUsage. */

 public int computeCPUUtilization(){
  return computeUsage(CPUUtilization);
 }


 /* Calculate page fault rate. This is done by calling a private
  * method -- computeUsage. */

 public int computePageFaultR(){
  return computeUsage(pagefaultRate);
 }


 /* Calculate throughput which is the number of processes that
  * finish execution during TIME_UNIT. */

 public synchronized int computeThroughput(long currentTime){
  // TIME_UNIT: milliseconds.
  Date startDate = new Date(currentTime - ConfigCurrent.TIME_UNIT);
  int NumFinishedProc = 0;
  Date finishDate;

  for (Enumeration e = throughput.elements() ; e.hasMoreElements() ;) {
    finishDate = new Date( ((Long) e.nextElement()).longValue());
    if (finishDate.after(startDate))
      NumFinishedProc++;
    else
      break;
  }
  return NumFinishedProc*60*1000/ConfigCurrent.TIME_UNIT; // Per minute.
 }


 /* Calculate turn around time. This is done by calling a private
  * method -- computeTime. */

 public int computeTurnaroundT(){
  return computeTime(turnaroundTime);
```

```
 }

 /* Calculate response time. This is done by calling a private
  * method -- computeTime. */

 public int computeResponseT(){
  return computeTime(responseTime);
 }

 /* Calculate waiting time. This is done by calling a private
  * method -- computeTime. */

 public int computeWaitingT(){
  return computeTime(waitingTime);
 }

 /* The parameter of this method is a vector of boolean value.
  * The method calculates the number of true values out of total
  * boolean values in the vector. */

 private synchronized int computeUsage(Vector pVector){
  boolean affirmative;
  int NumAffirmative = 0;

  for (Enumeration e = pVector.elements() ; e.hasMoreElements() ;) {
   affirmative = ((Boolean) e.nextElement()).booleanValue();
   if (affirmative)
    NumAffirmative++;
  }
  if (pVector.size() == 0)
   return 0;
  else
   return (NumAffirmative*100)/pVector.size();
 }

 /* The parameter of this method is a vector of integer values (times).
  * This method calculates the average values of these integers. */

 private synchronized int computeTime(Vector pVector){
  int pTime = 0;

  for (Enumeration e = pVector.elements() ; e.hasMoreElements() ;)
   pTime = pTime + ((Integer) e.nextElement()).intValue();

  if (pVector.size() == 0)
   return -1;
  else
   return (pTime/pVector.size())/1000; // Seconds.
 }

 /* Add an integer value (time) into a vector. When the number of integers
  * kept in the vector exceeds a maximum number, the oldest is removed. */

 private synchronized void addVector(Vector pVector, int pTime){
  pVector.insertElementAt(new Integer(pTime), 0);
  if (pVector.size() > ConfigCurrent.MAXPROCESS)
   pVector.removeElementAt(ConfigCurrent.MAXPROCESS);
 }

}
```

A.2. ConfigCurrent.java

```
import java.io.*;

/* Current system configuration values. */

public class ConfigCurrent{

 // Number of registers.
 public static int NREGS = 8;

 // Memory size in byte (better no less than 16).
 public static int MEMSIZE = 128;

 // Number of bytes in a page (or frame).
 public static int PAGESIZE = 2;

 // Number of pages per process.
 public static int NUMPAGES = 12;

 // Size of address space per process (virtual memory).
 public static int ADDRSPACESIZE = PAGESIZE * NUMPAGES;

 // Total number of memory frames.
 public static int MEMORYFRAMES = MEMSIZE / PAGESIZE;

 // Maximum number of processes (8, 16, 24).
 public static int MAXPROCESS = 8;

 // Size of backing-store.
 public static int BACKSTORESIZE = MAXPROCESS * NUMPAGES * PAGESIZE;

 // Page table entries.
 public static int PAGEENTRY = MAXPROCESS * NUMPAGES;

 // Quantum for round robin scheduling.
 public static int QUANTUM = 2;

 // Maximum number of memory access for a process (size of a process).
 public static int TOTAL_MEMORY_ACCESS = 24;

 // Thrashold to activate memory replacement algorithm.
 public static int FREE_FRAME_THRASHOLD = (MEMSIZE / PAGESIZE)/4;

 // Number of frames retrieved in FIFO memory replacement algorithm.
 public static int RETRIEVED_FRAMES = (MEMSIZE / PAGESIZE)/2;

 // Scheduling algorithm.
 public static int PROCESS_MANAGEMENT = EnumScheduling.ROUND_ROBIN;

 // Memory replacement algorithm.
 public static int MEMORY_MANAGEMENT = EnumPageReplacement.FIFO;

 // Priority level of processes.
 public static int PRIORITY_LEVEL = 6;

 // DMA delay.
 public static int DMADELAY = 2;

 // Number of keys regarding message communications.
 public static int NUMKEYS = 2;

 // Number of semaphores for resource one.
 public static int NUMSEM1 = 1;

 // Number of semaphores for resource two.
 public static int NUMSEM2 = 2;

 // Semaphore quantum (ticks).
```

```
public static int SEMQUANTUM = 2;

// Maximum number of execution history items.
public static int MAX_HISTORY_ITEMS = 600;

// Execution delay time (milliseconds).
public static int WAIT_TIME = 300;

// Execution delay time portion (milliseconds).
public static int WAIT_TIME_PORTION = 80;

// Minimum execution delay time (milliseconds).
public static int WAIT_TIME_MIN = 100;

// The period of time in which throughput is calculated (seconds).
public static int TIME_UNIT = 10000; //milliseconds = 10 seconds

// Number of pollings for cpu usage.
public static int MAX_CPU_USE_BITS = 64;

// Number of pollings for memory access.
public static int MAX_MEM_USE_BITS = 16;

// Simulation mode.
public static int MODE = EnumSimulationMode.INTERACTIVE;

// Indicate processes contain memory operations or not.
public static boolean MEMORY_OPERATION = true;

// Indicate processes contain message operations or not.
public static boolean MESSAGE_OPERATION = true;

// Indicate processes contain semaphore operations or not.
public static boolean SEMAPHORE_OPERATION = true;

// Indicate whether print out information to a file or not.
public static boolean OUTPUT_TO_FILE = false;

// File name for printing out information.
public static String OUTPUT_FILE_NAME = "output.sim";

// Indicate whether print out system configuration information.
public static boolean OUTPUT_SYSTEM_INF = false;

// Indicate whether print out system performance information.
public static boolean OUTPUT_RESULT_INF = false;

// Indicate whether print out execution history information.
public static boolean OUTPUT_HISTORY_INF = false;


/* Print out system configuration information */

public static void printout(PrintStream os) throws IOException{
  IOFormat.print(os, "%s\n", "*****************************************************");
  IOFormat.print(os, "%s\n", "*      SYSTEM CONFIGURATION INFORMATION       *");
  IOFormat.print(os, "%s\n", "*****************************************************");

  output(os, "Register numbers ---- ", NREGS);
  output(os, "Memory size in byte ---- ", MEMSIZE);
  output(os, "Number of bytes in a page (or frame) ---- ", PAGESIZE);
  output(os, "Number of pages per process ", NUMPAGES);
  output(os, "Size of address space per process (virtual memory) ---- ", ADDRSPACESIZE);
  output(os, "Total number of memory frames in system ---- ", MEMORYFRAMES);
  output(os, "Maximum number of processes ---- ", MAXPROCESS);
  output(os, "Backstore size in byte ---- ", BACKSTORESIZE);
  output(os, "Number of page table entries ---- ", PAGEENTRY);
  output(os, "Quantum for round robin scheduling ---- ", QUANTUM);
  output(os, "Maximum memory access number per process ---- ", TOTAL_MEMORY_ACCESS);
  output(os, "Thrashold to activate memory replacement algorithm ---- ", FREE_FRAME_THRASHOLD);
```

```
    output(os, "Number of frames retrieved in FIFO memory replacement algorithm ---- ", RETRIEVED_FRAMES);

    if(PROCESS_MANAGEMENT == EnumScheduling.ROUND_ROBIN)
      output(os, "Scheduling algorithm ---- ", "Round robin");
    else
      output(os, "Scheduling algorithm ---- ", "Priority");

    if(MEMORY_MANAGEMENT == EnumPageReplacement.FIFO)
      output(os, "Memory replacement algorithm ---- ", "FIFO");
    else
      output(os, "Memory replacement algorithm ---- ", "Second chance");

    output(os, "Priority level of processes ---- ", PRIORITY_LEVEL);
    output(os, "DMA delay (ticks) ---- ", DMADELAY);
    output(os, "Number of message communication keys ---- ", NUMKEYS);
    output(os, "Number of semaphores for resource one ---- ", NUMSEM1);
    output(os, "Number of semaphores for resource two ---- ", NUMSEM2);
    output(os, "Semaphore quantum (ticks) ---- ", SEMQUANTUM);
    output(os, "Maximum number of execution history items ---- ", MAX_HISTORY_ITEMS);
    output(os, "Execution delay time (milliseconds) ---- ", WAIT_TIME);
    output(os, "Execution delay time portion (milliseconds) ---- ", WAIT_TIME_PORTION);
    output(os, "Minimum execution delay time (milliseconds) ---- ", WAIT_TIME_MIN);
    output(os, "The period of time in which throughput is calculated (seconds) ---- ", TIME_UNIT);
    output(os, "Number of pollings for cpu usage ---- ", MAX_CPU_USE_BITS);
    output(os, "Number of pollings for memory access ---- ", MAX_MEM_USE_BITS);

    if (MODE == EnumSimulationMode.INTERACTIVE)
      output(os, "Simulation mode ---- ", "interactive");
    if (MODE == EnumSimulationMode.NON_INTERACTIVE)
      output(os, "Simulation mode ---- ", "none interactive");

    if (MEMORY_OPERATION)
      output(os, "Processes contain memory operations", "");
    if (MESSAGE_OPERATION)
      output(os, "Processes contain message operations", "");
    if (SEMAPHORE_OPERATION)
      output(os, "Processes contain semaphore operations", "");

    if (OUTPUT_TO_FILE)
      output(os, "output file name ---- ", OUTPUT_FILE_NAME);

    if (OUTPUT_SYSTEM_INF)
      output(os, " print out system configuration information", "");
    if (OUTPUT_RESULT_INF)
      output(os, " print out system performance information", "");
    if (OUTPUT_HISTORY_INF)
      output(os, " print out execution history information", "");

    IOFormat.print(os, "%s\n", " ");
  }


  /* Print out a configuration information line by name and value. */

  private static void output(PrintStream os, String name, int value){
    IOFormat.print(os, "%s", name);
    IOFormat.print(os, "%d\n", value);
  }


  /* Print out a configuration information line by name and boolean value. */

  private static void output(PrintStream os, String name, boolean b){
    IOFormat.print(os, "%s", name);
    if ( b )
      IOFormat.print(os, "%s\n", "true");
    else
      IOFormat.print(os, "%s\n", "false");
  }
```

```
 /* Print out a configuration information line by name and string value. */

 private static void output(PrintStream os, String name, String str){
   IOFormat.print(os, "%s", name);
   IOFormat.print(os, "%s\n", str);
 }


}
```

A.3. ConfigDefault.java

```
/* Default system configuration values. */

public class ConfigDefault{

 // Number of registers.
 public static int NREGS = 8;

 // Memory size in byte (better no less than 16).
 public static int MEMSIZE = 128;

 // Number of bytes in a page (or frame).
 public static int PAGESIZE = 2;

 // Number of pages per process.
 public static int NUMPAGES = 12;

 // Size of address space per process (virtual memory).
 public static int ADDRSPACESIZE = PAGESIZE * NUMPAGES;

 // Total number of memory frames.
 public static int MEMORYFRAMES = MEMSIZE / PAGESIZE;

 // Maximum number of processes (8, 16, 24).
 public static int MAXPROCESS = 8;

 // Size of backstore.
 public static int BACKSTORESIZE = MAXPROCESS * NUMPAGES * PAGESIZE;

 // Page table entries.
 public static int PAGEENTRY = MAXPROCESS * NUMPAGES;

 // Quantum for round robin scheduling.
 public static int QUANTUM = 2;

 // Maximum number of memory access for a process (size of a process).
 public static int TOTAL_MEMORY_ACCESS = 24;

 // Thrashold to activate memory replacement algorithm.
 public static int FREE_FRAME_THRASHOLD = (MEMSIZE / PAGESIZE)/4;

 // Number of frames retrieved in FIFO memory replacement algorithm.
 public static int RETRIEVED_FRAMES = (MEMSIZE / PAGESIZE)/2;

 // Scheduling algorithm.
 public static int PROCESS_MANAGEMENT = EnumScheduling.ROUND_ROBIN;

 // Memory replacement algorithm.
 public static int MEMORY_MANAGEMENT = EnumPageReplacement.FIFO;

 // Priority level of processes.
 public static int PRIORITY_LEVEL = 6;

 // DMA delay.
 public static int DMADELAY = 2;

 // Number of keys regarding message communications.
 public static int NUMKEYS = 2;

 // Number of semaphores for resource one.
 public static int NUMSEM1 = 1;

 // Number of semaphores for resource two.
 public static int NUMSEM2 = 2;

 // Semaphore quantum (ticks).
 public static int SEMQUANTUM = 2;
```

```
// Maximum number of execution history items.
public static int MAX_HISTORY_ITEMS = 600;

// Execution delay time (milliseconds).
public static int WAIT_TIME = 300;

// Execution delay time portion (milliseconds).
public static int WAIT_TIME_PORTION = 80;

// Minimum execution delay time (milliseconds).
public static int WAIT_TIME_MIN = 100;

// The period of time in which throughput is calculated (seconds).
public static int TIME_UNIT = 10000; //milliseconds = 10 seconds

// Number of pollings for cpu usage.
public static int MAX_CPU_USE_BITS = 64;

// Number of pollings for memory access.
public static int MAX_MEM_USE_BITS = 16;

// Simulation mode.
public static int MODE = EnumSimulationMode.INTERACTIVE;

// Indicate processes contain memory operations or not.
public static boolean MEMORY_OPERATION = true;

// Indicate processes contain message operations or not.
public static boolean MESSAGE_OPERATION = true;

// Indicate processes contain semaphore operations or not.
public static boolean SEMAPHORE_OPERATION = true;

// Indicate whether print out information to a file or not.
public static boolean OUTPUT_TO_FILE = false;

// File name for printing out information.
public static String OUTPUT_FILE_NAME = "output.sim";

// Indicate whether print out system configuration information.
public static boolean OUTPUT_SYSTEM_INF = false;

// Indicate whether print out system performance information.
public static boolean OUTPUT_RESULT_INF = false;

// Indicate whether print out execution history information.
public static boolean OUTPUT_HISTORY_INF = false;

}
```

A.4. Configuration.java

```
import java.awt.*;
import java.io.*;
import java.util.*;


/* class Configuration is responsible for setting the system configuration.
 * The system configuration contains those values on which the simulation
 * is based. The user can change these values in reasonable ranges and also
 * can set these values to the default values. */

public class Configuration extends Dialog {

 private Integer cf;       // Newly defined value of page size in bytes.
 private Integer de;       // Default value of page size in bytes.
 private Panel cards;      // Layout area of configuration values.
 private Panel tabs;       // Layout area of configuration catalog buttons.
 private Panel btnbar;     // Layout area of action buttonsr.
 private CardLayout layout;
 private Button btnS;      // For switching to values of system.
 private Button btnMM;     // For switching to values of memory management.
 private Button btnPM;     // For switching to values of process management.
 private Button btnIPC;    // For switching to values of IPC.
 private Button btnSM;     // For switching to values of simulation mode.
 private Button btnOR;     // For switching to values of output result.
 private Button btnOK;     // For commiting newly defined values.
 private Button btnCANCEL; // For canceling newly defined values and
                           // Closing the configuration window.
 private Button btnDEFAULT; // For setting to default valuers.

 private Panel pnlS;       // For layout values of system.
 private Panel pnlMM;      // For layout values of memory management.
 private Panel pnlPM;      // For layout values of process management.
 private Panel pnlIPC;     // For layout values of IPC.
 private Panel pnlSM;      // For layout values of simulation mode.
 private Panel pnlOR;      // For layout values of output result.
 private Panel currentpnl; // Indicating which panel is currently displayed.

 private Choice mchoice;   // Choice for memory size in bytes.
 private Choice pchoice;   // Choice for page size in bytes.
 private Choice pnchoice;  // Choice for number of pages per process.
 private Choice mpchoice;  // Choice for maximum number of processes.
 private Choice qchoice;   // Choice for quantum for round robin scheduling.
 private Choice mmchoice;  // Choice for maximum number of memory access
                           // for a processr.
 private Choice plchoice;  // Choice for priority level of processes.
 private Choice dchoice;   // Choice for DMA delay.
 private Choice mckchoice; // Choice for number of keys regarding.
                           // message communications.
 private Choice s1choice;  // Choice for number of semaphore one.
 private Choice s2choice;  // Choice for number of semaphore two.
 private Choice sqchoice;  // Choice for semaphore quantum for
                           // shared resources usage.
 private Choice mhchoice;  // Choice for maximum number of items of history.
 private Choice wtchoice;  // Choice for execution delay time (milliseconds).
 private Choice wtpchoice; // Choice for execution delay time portion
                           // (milliseconds).
 private Choice wtmchoice; // Choice for minimum execution delay time
                           // (milliseconds).
 private Choice tuchoice;  // Choice for time period in which throughput
                           // is calculated.
 private Choice mcchoice;  // Choice for number of pollings for cpu usage.
 private Choice mmuchoice; // Choice for number of pollings for memory access

 private Label mlabel;     // Label for memory size in bytes.
 private Label plabel;     // Label for page size in bytes.
 private Label pnlabel;    // Label for number of pages per process.
```

```
private Label mplabel;     // Label for maximum number of processes.
private Label qlabel;      // Label for quantum for round robin scheduling.
private Label mmlabel;     // Label for maximum number of memory access
                           // for a process.
private Label pllabel;     // Label for priority level of processes.
private Label dlabel;      // Label for DMA delay.
private Label mcklabel;    // Label for number of keys regarding
                           // message communications.
private Label s1label;     // Label for number of semaphore one.
private Label s2label;     // Label for number of semaphore two.
private Label sqlabel;     // Label for semaphore quantum for
                           // shared resources usage.
private Label mhlabel;     // Label for maximum number of items of history.
private Label wtlabel;     // Label for execution delay time (milliseconds).
private Label wtplabel;    // Label for execution delay time portion
                           // (milliseconds).
private Label wtmlabel;    // Label for minimum execution delay time.
private Label tulabel;     // Label for time period in which throughput
                           // is calculated.
private Label mclabel;     // Label for number of pollings for cpu usage.
private Label mmulabel;    // Label for number of pollings for memory access

private Label asslabel;    // Label for address space size.
private Label asslabel_v;  // Value field for address space size.
private Label tmflabel;    // Label for total memory frame numbers.
private Label tmflabel_v;  // Value field for total memory frame numbers.
private Label bslabel;     // Label for backstore size.
private Label bslabel_v;   // Value field for backstore size.
private Label pelabel;     // Label for page entry numbers.
private Label pelabel_v;   // Value field for page entry numbers.
private Label tfflabel;    // Label for thrashold for free frame list.
private Label tfflabel_v;  // Value field for thrashold for free frame list.
private Label rfnlabel;    // Label for retrieved frame numbers.
private Label rfnlabel_v;  // Value field for retrieved frame numbers.
private Label rnlabel;     // Label for register numbers.
private Label rnlabel_v;   // Value field for register numbers.

private Label pmlabel;     // Label for scheduling algorithm.
private Choice p_manage;   // Choice for scheduling algorithm.
private Label memlabel;    // Label for page replacement algorithm.
private Choice m_manage;   // Choice for page replacement algorithm.


private Label smlabel;     // Label for simulation mode.
private Label ptlabel;     // Label for process operation types.
private Checkbox sm_check1; // "Interactive" checkbox for simulation mode.
private Checkbox sm_check2; // "Non_Interactive" checkbox for simulation mode.
private Checkbox pt_check1; // "Memory" checkbox for process operation types.
private Checkbox pt_check2; // "Message" checkbox for process operation types.
private Checkbox pt_check3; // "Semaphore" checkbox for process
                            // operation types.

private Label otflabel;    // Label for output to file.
private Label crlabel;     // Label for contents of output.
private Checkbox otf_check1; // "Yes" checkbox for output to file.
private Checkbox otf_check2; // "No" checkbox for output to file.
private Checkbox cr_check1; // "System information" checkbox for
                            // contents of output.
private Checkbox cr_check3; // "Simulation result" checkbox for
                            // contents of output.
private Checkbox cr_check4; // "Execution history" checkbox for
                            // contents of output.
private Label fnlabel;     // Label for file name.
private TextField fntext;  // Text field for file name.
private Button fnbutton;   // Button for pop up a window for
                           // selecting a file name.
private Frame p;           // Main user interface window.
private FileDialog fd;     // Dialog window for output file name selection.
```

```
/* The constructor initializes Configuration. */

public Configuration(Frame parent) {
 super(parent, true);
 setTitle("Configuration");

 readData(); // Generate the current system configuration from Config.currentfile.

 tabs = new Panel();
 GridLayout gl = new GridLayout(1, 6);
 tabs.setLayout(gl);
 btnS = new Button("System");
 btnS.setForeground(Color.black);
 btnS.setBackground(new Color(246,244,249));
 tabs.add(btnS);
 btnMM = new Button("Memory Management");
 btnMM.setForeground(Color.black);
 btnMM.setBackground(new Color(225,223,227));
 tabs.add(btnMM);
 btnPM = new Button("Process Management");
 btnPM.setForeground(Color.black);
 btnPM.setBackground(new Color(203,201,206));
 tabs.add(btnPM);
 btnIPC = new Button("IPC");
 btnIPC.setForeground(Color.black);
 btnIPC.setBackground(new Color(178,176,181));
 tabs.add(btnIPC);
 btnSM = new Button("Simulation Mode");
 btnSM.setForeground(Color.black);
 btnSM.setBackground(new Color(157,156,160));
 tabs.add(btnSM);
 btnOR = new Button("Output Results");
 btnOR.setForeground(Color.black);
 btnOR.setBackground(new Color(142,140,144));
 tabs.add(btnOR);
 add("North", tabs);

 cards = new Panel();
 layout = new CardLayout();
 cards.setLayout(layout);

 cards.add("System", pnlS = new Panel());
 currentpnl = pnlS;
 pnlS.setForeground(Color.black);
 pnlS.setBackground(new Color(246,244,249));
 setPnlS(); // Set up a configuration panel for System.
 cards.add("Memory Management", pnlMM = new Panel());
 pnlMM.setForeground(Color.black);
 pnlMM.setBackground(new Color(225,223,227));
 setPnlMM(); // Set up a configuration panel for Memory Management.
 cards.add("Process Management", pnlPM = new Panel());
 pnlPM.setForeground(Color.black);
 pnlPM.setBackground(new Color(203,201,206));
 setPnlPM(); // Set up a configuration panel for Process Management.
 cards.add("IPC", pnlIPC = new Panel());
 pnlIPC.setForeground(Color.black);
 pnlIPC.setBackground(new Color(178,176,181));
 setPnlIPC(); // Set up a configuration panel for IPC.
 cards.add("Simulation Mode", pnlSM = new Panel());
 pnlSM.setForeground(Color.black);
 pnlSM.setBackground(new Color(157,156,160));
 setPnlSM(); // Set up a configuration panel for Simulation Mode.
 cards.add("Output Results", pnlOR = new Panel());
 pnlOR.setForeground(Color.black);
 pnlOR.setBackground(new Color(142,140,144));
 setPnlOR(); //  Set up a configuration panel for Output Results.

 add("Center", cards);
 btnbar = new Panel();
 btnOK = new Button("OK");
```

```
  btnbar.add(btnOK);
  btnCANCEL = new Button("CANCEL");
  btnbar.add(btnCANCEL);
  btnDEFAULT = new Button("DEFAULTS");
  btnbar.add(btnDEFAULT);
  add("South", btnbar);

  p = parent;
 }


 /* Show different configuration panel according to user's click on
  * configuration catalog buttons. */

 public boolean handleEvent(Event event) {
  if (event.id == Event.WINDOW_DESTROY) {
    System.exit(0);
    return super.handleEvent(event);
  }
  else if(event.target == btnS && event.id == Event.END) {
    layout.show(cards, btnS.getLabel());
    currentpnl = pnlS;
    return(true);
  }
  else if(event.target == btnMM && event.id == Event.END) {
    layout.show(cards, btnMM.getLabel());
    currentpnl = pnlMM;
    return(true);
  }
  else if(event.target == btnPM && event.id == Event.END) {
    layout.show(cards, btnPM.getLabel());
    currentpnl = pnlPM;
    return(true);
  }
  else if(event.target == btnIPC && event.id == Event.END) {
    layout.show(cards, btnIPC.getLabel());
    currentpnl = pnlIPC;
    return(true);
  }
  else if(event.target == btnSM && event.id == Event.END) {
    layout.show(cards, btnSM.getLabel());
    currentpnl = pnlSM;
    return(true);
  }
  else if(event.target == btnOR && event.id == Event.END) {
    layout.show(cards, btnOR.getLabel());
    currentpnl = pnlOR;
    return(true);
  }
  else
    return super.handleEvent(event);
 }


 /* When user changes a configuration value, some dependent configuration
  * values will change accordingly if any. User can choose OK or Cancel to
  * make the newly set configuration values take effect or not; User can
  * also choose DEFAULT to set configuration values to default values. */

 public boolean action(Event evt, Object arg) {
  int i1, i2, i3, i4;
  Integer In;

  i1 = IOFormat.atoi(mchoice.getSelectedItem());
  i2 = IOFormat.atoi(pchoice.getSelectedItem());
  i3 = IOFormat.atoi(pnchoice.getSelectedItem());
  i4 = IOFormat.atoi(mpchoice.getSelectedItem());

  if (evt.target.equals(btnOK)) {
```

```
    // Set the partial current system configuration by the vaules on
    // configuration window.
    setConfig();

    // Set the partial current system configuration by some dependented
    // values of the current system configuration.
    calculateData();

    // Before disposing the configuration window, set the configuration
    // panel for the next time.
    layout.first(cards);
    currentpnl = pnlS;

    printConfig(); // Print current system configuration to standard output.
    dispose(); // dispose the configuration window.
  }
  else if (evt.target.equals(btnDEFAULT))
    setDefault(); // Set system configuration to default values.
  else if (evt.target.equals(btnCANCEL)) {

    // Set the system configuration to current system configuration,
    // ignore the newly user-defined value.
    setCancel();

    // Before disposing the configuration window, set the configuration
    // panel for the next time.
    layout.first(cards);
    currentpnl = pnlS;

    dispose(); // Dispose the configuration window.
    return super.action(evt, arg);
  }

  // When user changes a configuration value, some dependent configuration
  // values will change accordingly if any.
  else if (evt.target.equals(mchoice)) {
    In = new Integer(i1/i2);
    tmflabel_v.setText(In.toString());

    In = new Integer(i1/i2/4);
    tfflabel_v.setText(In.toString());

    In = new Integer(i1/i2/2);
    rfnlabel_v.setText(In.toString());
  }
  else if (evt.target.equals(pchoice)) {
    In = new Integer(i2*i3);
    asslabel_v.setText(In.toString());

    In = new Integer(i1/i2);
    tmflabel_v.setText(In.toString());

    In = new Integer(i4*i3*i2);
    bslabel_v.setText(In.toString());

    In = new Integer(i1/i2/4);
    tfflabel_v.setText(In.toString());

    In = new Integer(i1/i2/2);
    rfnlabel_v.setText(In.toString());
  }
  else if (evt.target.equals(pnchoice)) {
    In = new Integer(i2*i3);
    asslabel_v.setText(In.toString());

    In = new Integer(i4*i3*i2);
    bslabel_v.setText(In.toString());

    In = new Integer(i4*i3);
    pelabel_v.setText(In.toString());
```

```
    }
    else if (evt.target.equals(mpchoice)) {
      In = new Integer(i4*i3*i2);
      bslabel_v.setText(In.toString());

      In = new Integer(i4*i3);
      pelabel_v.setText(In.toString());
    }
    else if (evt.target.equals(fnbutton)) {
      fd = new FileDialog(p, "Select file");
      p.add(fd);
      fd.show(); // Show "select file" dialog.
      String fn = fd.getFile();
      if (fn.length() == 0)
        fntext.setText(ConfigDefault.OUTPUT_FILE_NAME);
      else
        fntext.setText(fd.getFile());
    }
    else if (evt.target.equals(otf_check1))
      pnlOREn(); // Enable partial of the configuration panel for Output Results.
    else if (evt.target.equals(otf_check2))
      pnlORDis(); // Disable partial of the configuration panel for Output Results.
    else if (evt.target.equals(sm_check1))
      pnlSMDis(); // Enable partial of the configuration panel for Simulation Modes.
    else if (evt.target.equals(sm_check2))
      pnlSMEn(); // Disable partial of the configuration panel for Simulation Modes.
    else return super.action(evt, arg);
      return true;
  }


  /* Set up a configuration panel for System. */

  private void setPnlS(){
    int none = GridBagConstraints.NONE;
    int hori = GridBagConstraints.HORIZONTAL;
    int vert = GridBagConstraints.VERTICAL;
    int both = GridBagConstraints.BOTH;
    int center = GridBagConstraints.CENTER;
    int north = GridBagConstraints.NORTH;
    int northeast = GridBagConstraints.NORTHEAST;
    int east = GridBagConstraints.EAST;
    int west = GridBagConstraints.WEST;
    GridBagLayout gbl = new GridBagLayout();
    pnlS.setLayout(gbl);
    GridBagConstraints gbc = new GridBagConstraints();

    plabel = new Label("Page size:");
    cf = new Integer(ConfigCurrent.PAGESIZE);
    de = new Integer(ConfigDefault.PAGESIZE);
    pchoice = new Choice();
    pchoice.addItem(de.toString());
    pchoice.addItem("4");
    pchoice.addItem("6");
    pchoice.select(cf.toString());

    dlabel = new Label("DMA delay:");
    cf = new Integer(ConfigCurrent.DMADELAY);
    de = new Integer(ConfigDefault.DMADELAY);
    dchoice = new Choice();
    dchoice.addItem(de.toString());
    dchoice.addItem("4");
    dchoice.addItem("6");
    dchoice.select(cf.toString());

    mhlabel = new Label("Maximum history items:");
    cf = new Integer(ConfigCurrent.MAX_HISTORY_ITEMS);
    de = new Integer(ConfigDefault.MAX_HISTORY_ITEMS);
    mhchoice = new Choice();
    mhchoice.addItem("200");
```

```
mhchoice.addItem("400");
mhchoice.addItem(de.toString());
mhchoice.select(cf.toString());

wtlabel = new Label("Wait time:");
cf = new Integer(ConfigCurrent.WAIT_TIME);
de = new Integer(ConfigDefault.WAIT_TIME);
wtchoice = new Choice();
wtchoice.addItem("100");
wtchoice.addItem(de.toString());
wtchoice.addItem("500");
wtchoice.select(cf.toString());

wtplabel = new Label("Wait time portion:");
cf = new Integer(ConfigCurrent.WAIT_TIME_PORTION);
de = new Integer(ConfigDefault.WAIT_TIME_PORTION);
wtpchoice = new Choice();
wtpchoice.addItem("40");
wtpchoice.addItem("60");
wtpchoice.addItem(de.toString());
wtpchoice.select(cf.toString());

wtmlabel = new Label("Wait time min:");
cf = new Integer(ConfigCurrent.WAIT_TIME_MIN);
de = new Integer(ConfigDefault.WAIT_TIME_MIN);
wtmchoice = new Choice();
wtmchoice.addItem(de.toString());
wtmchoice.addItem("150");
wtmchoice.select(cf.toString());

tulabel = new Label("Time unit:");
cf = new Integer(ConfigCurrent.TIME_UNIT);
de = new Integer(ConfigDefault.TIME_UNIT);
tuchoice = new Choice();
tuchoice.addItem(de.toString());
tuchoice.addItem("50000");
tuchoice.addItem("100000");
tuchoice.select(cf.toString());

mclabel = new Label("Max cpu use bits:");
cf = new Integer(ConfigCurrent.MAX_CPU_USE_BITS);
de = new Integer(ConfigDefault.MAX_CPU_USE_BITS);
mcchoice = new Choice();
mcchoice.addItem("32");
mcchoice.addItem(de.toString());
mcchoice.addItem("128");
mcchoice.addItem("640");
mcchoice.addItem("1280");
mcchoice.select(cf.toString());

mmulabel = new Label("Max mem use bits:");
cf = new Integer(ConfigCurrent.MAX_MEM_USE_BITS);
de = new Integer(ConfigDefault.MAX_MEM_USE_BITS);
mmuchoice = new Choice();
mmuchoice.addItem(de.toString());
mmuchoice.addItem("32");
mmuchoice.addItem("64");
mmuchoice.addItem("128");
mmuchoice.addItem("640");
mmuchoice.addItem("1280");
mmuchoice.select(cf.toString());

mlabel = new Label("Register numbers:");
cf = new Integer(ConfigCurrent.NREGS);
mlabel_v = new Label(cf.toString());

bslabel = new Label("Backstore size:");
cf = new Integer(ConfigCurrent.MAXPROCESS * ConfigCurrent.NUMPAGES *
        ConfigCurrent.PAGESIZE);
bslabel_v = new Label(cf.toString());
```

```
  pelabel = new Label("Page entry numbers:");
  cf = new Integer(ConfigCurrent.MAXPROCESS * ConfigCurrent.NUMPAGES);
  pelabel_v = new Label(cf.toString());

  Label dumy1 = new Label("  ");
  Label dumy2 = new Label("  ");

  add(pnlS, plabel, gbl, gbc, 0, 0, 1, 1, none, west, 0, 0);
  add(pnlS, pchoice, gbl, gbc, 1, 0, 1, 1, none, west, 0, 0);
  add(pnlS, dlabel, gbl, gbc, 0, 1, 1, 1, none, west, 0, 0);
  add(pnlS, dchoice, gbl, gbc, 1, 1, 1, 1, none, west, 0, 0);
  add(pnlS, mhlabel, gbl, gbc, 0, 2, 1, 1, none, west, 0, 0);
  add(pnlS, mhchoice, gbl, gbc, 1, 2, 1, 1, none, west, 0, 0);
  add(pnlS, mclabel, gbl, gbc, 0, 3, 1, 1, none, west, 0, 0);
  add(pnlS, mcchoice, gbl, gbc, 1, 3, 1, 1, none, west, 0, 0);
  add(pnlS, mmulabel, gbl, gbc, 0, 4, 1, 1, none, west, 0, 0);
  add(pnlS, mmuchoice, gbl, gbc, 1, 4, 1, 1, none, west, 0, 0);

  add(pnlS, dumy1, gbl, gbc, 2, 0, 1, 5, none, center, 0, 0);

  add(pnlS, wtlabel, gbl, gbc, 3, 3, 1, 1, none, west, 0, 0);
  add(pnlS, wtchoice, gbl, gbc, 4, 3, 1, 1, none, west, 0, 0),
  add(pnlS, wtplabel, gbl, gbc, 3, 1, 1, 1  none, west, 0, 0);
  add(pnlS, wtpchoice, gbl, gbc, 4, 1, 1, 1, none, west, 0, 0);
  add(pnlS, wtmlabel, gbl, gbc, 3, 2, 1, 1, none, west, 0, 0);
  add(pnlS, wtmchoice, gbl, gbc, 4, 2, 1, 1, none, west, 0, 0);
  add(pnlS, tulabel, gbl, gbc, 3, 0, 1, 1, none, west, 0, 0);
  add(pnlS, tuchoice, gbl, gbc, 4, 0, 1, 1, none, west, 0, 0);

  add(pnlS, dumy2, gbl, gbc, 5, 0, 1, 5, none, center, 0, 0);

  add(pnlS, bslabel, gbl, gbc, 6, 0, 1, 1, none, west, 0, 0);
  add(pnlS, bslabel_v, gbl, gbc, 7, 0, 1, 1, none, west, 0, 0);
  add(pnlS, pelabel, gbl, gbc, 6, 1, 1, 1, none, west, 0, 0);
  add(pnlS, pelabel_v, gbl, gbc, 7, 1, 1, 1, none, west, 0, 0);
  add(pnlS, mlabel, gbl, gbc, 6, 2, 1, 1, none, west, 0, 0);
  add(pnlS, mlabel_v, gbl, gbc, 7, 2, 1, 1, none, west, 0, 0);
}


/* Set up a configuration panel for Memory Management. */

private void setPnlMM(){
  int none = GridBagConstraints.NONE;
  int hori = GridBagConstraints.HORIZONTAL;
  int vert = GridBagConstraints.VERTICAL;
  int both = GridBagConstraints.BOTH;
  int center = GridBagConstraints.CENTER;
  int north = GridBagConstraints.NORTH;
  int northeast = GridBagConstraints.NORTHEAST;
  int east = GridBagConstraints.EAST;
  int west = GridBagConstraints.WEST;

  GridBagLayout gbl = new GridBagLayout();
  pnlMM.setLayout(gbl);
  GridBagConstraints gbc = new GridBagConstraints();

  mlabel = new Label("Memory size:");
  cf = new Integer(ConfigCurrent.MEMSIZE);
  de = new Integer(ConfigDefault.MEMSIZE);
  mchoice = new Choice();
  mchoice.addItem("64");
  mchoice.addItem(de.toString());
  mchoice.addItem("256");
  mchoice.addItem("640");
  mchoice.select(cf.toString());

  mmlabel = new Label("Max. MA numbers per process:");
  cf = new Integer(ConfigCurrent.TOTAL_MEMORY_ACCESS);
```

```
  de = new Integer(ConfigDefault.TOTAL_MEMORY_ACCESS);
  mmchoice = new Choice();
  mmchoice.addItem(de.toString());
  mmchoice.addItem("64");
  mmchoice.addItem("128");
  mmchoice.select(cf.toString());

  memlabel = new Label("Page replacement algorithm:");
  m_manage = new Choice();
  m_manage.addItem("FIFO");
  m_manage.addItem("SECOND CHANCE");
  if(ConfigCurrent.MEMORY_MANAGEMENT == EnumPageReplacement.FIFO)
    m_manage.select("FIFO");
  else
    m_manage.select("SECOND CHANCE");

  tmflabel = new Label("Total memory frame numbers:");
  cf = new Integer(ConfigCurrent.MEMSIZE / ConfigCurrent.PAGESIZE);
  tmflabel_v = new Label(cf.toString());

  tfflabel = new Label("Thrashold for free frame list:");
  cf = new Integer((ConfigCurrent.MEMSIZE / ConfigCurrent.PAGESIZE)/4);
  tfflabel_v = new Label(cf.toString());

  rfnlabel = new Label("Retrieved frame numbers:");
  cf = new Integer((ConfigCurrent.MEMSIZE / ConfigCurrent.PAGESIZE)/2);
  rfnlabel_v = new Label(cf.toString());

  Label dumy1 = new Label("  ");

  add(pnlMM, mlabel, gbl, gbc, 0, 0, 1, 1, none, west, 0, 0);
  add(pnlMM, mchoice, gbl, gbc, 1, 0, 1, 1, none, west, 0, 0);
  add(pnlMM, mmlabel, gbl, gbc, 0, 1, 1, 1, none, west, 0, 0);
  add(pnlMM, mmchoice, gbl, gbc, 1, 1, 1, 1, none, west, 0, 0);
  add(pnlMM, memlabel, gbl, gbc, 0, 2, 1, 1, none, west, 0, 0);
  add(pnlMM, m_manage, gbl, gbc, 1, 2, 1, 1, none, west, 0, 0);

  add(pnlMM, dumy1, gbl, gbc, 2, 0, 1, 3, none, center, 0, 0);

  add(pnlMM, tmflabel, gbl, gbc, 3, 0, 1, 1, none, west, 0, 0);
  add(pnlMM, tmflabel_v, gbl, gbc, 4, 0, 1, 1, none, west, 0, 0);
  add(pnlMM, tfflabel, gbl, gbc, 3, 1, 1, 1, none, west, 0, 0);
  add(pnlMM, tfflabel_v, gbl, gbc, 4, 1, 1, 1, none, west, 0, 0);
  add(pnlMM, rfnlabel, gbl, gbc, 3, 2, 1, 1, none, west, 0, 0);
  add(pnlMM, rfnlabel_v, gbl, gbc, 4, 2, 1, 1, none, west, 0, 0);
}


/* Set up a configuration panel for Process Management. */

private void setPnlPM(){
  int none = GridBagConstraints.NONE;
  int hori = GridBagConstraints.HORIZONTAL;
  int vert = GridBagConstraints.VERTICAL;
  int both = GridBagConstraints.BOTH;
  int center = GridBagConstraints.CENTER;
  int north = GridBagConstraints.NORTH;
  int northeast = GridBagConstraints.NORTHEAST;
  int east = GridBagConstraints.EAST;
  int west = GridBagConstraints.WEST;

  GridBagLayout gbl = new GridBagLayout();
  pnlPM.setLayout(gbl);
  GridBagConstraints gbc = new GridBagConstraints();

  pnlabel = new Label("Pages per process:");
  cf = new Integer(ConfigCurrent.NUMPAGES);
  de = new Integer(ConfigDefault.NUMPAGES);
  pnchoice = new Choice();
  pnchoice.addItem(de.toString());
```

```
  pnchoice.addItem("24");
  pnchoice.addItem("48");
  pnchoice.select(cf.toString());

  mplabel = new Label("Max. process numbers:");
  cf = new Integer(ConfigCurrent.MAXPROCESS);
  de = new Integer(ConfigDefault.MAXPROCESS);
  mpchoice = new Choice();
  mpchoice.addItem(de.toString());
  mpchoice.addItem("16");
  mpchoice.addItem("24");
  mpchoice.addItem("64");
  mpchoice.select(cf.toString());

  qlabel = new Label("Quantum for RR:");
  cf = new Integer(ConfigCurrent.QUANTUM);
  de = new Integer(ConfigDefault.QUANTUM);
  qchoice = new Choice();
  qchoice.addItem(de.toString());
  qchoice.addItem("8");
  qchoice.addItem("16");
  qchoice.select(cf.toString());

  pllabel = new Label("Priority level:");
  cf = new Integer(ConfigCurrent.PRIORITY_LEVEL);
  de = new Integer(ConfigDefault.PRIORITY_LEVEL);
  plchoice = new Choice();
  plchoice.addItem("2");
  plchoice.addItem("4");
  plchoice.addItem(de.toString());
  plchoice.select(cf.toString());

  pmlabel = new Label("Scheduling algorithm:");
  p_manage = new Choice();
  p_manage.addItem("ROUND ROBIN");
  p_manage.addItem("PRIORITY");
  if(ConfigCurrent.PROCESS_MANAGEMENT == EnumScheduling.ROUND_ROBIN)
    p_manage.select("ROUND ROBIN");
  else
    p_manage.select("PRIORITY");

  asslabel = new Label("Address space size:");
  cf = new Integer(ConfigCurrent.PAGESIZE * ConfigDefault.NUMPAGES);
  asslabel_v = new Label(cf.toString());

  Label dumy1 = new Label("    ");

  add(pnlPM, qlabel, gbl, gbc, 0, 0, 1, 1, none, west, 0, 0);
  add(pnlPM, qchoice, gbl, gbc, 1, 0, 1, 1, none, west, 0, 0);
  add(pnlPM, pnlabel, gbl, gbc, 0, 1, 1, 1, none, west, 0, 0);
  add(pnlPM, pnchoice, gbl, gbc, 1, 1, 1, 1, none, west, 0, 0);
  add(pnlPM, mplabel, gbl, gbc, 0, 2, 1, 1, none, west, 0, 0);
  add(pnlPM, mpchoice, gbl, gbc, 1, 2, 1, 1, none, west, 0, 0);
  add(pnlPM, pmlabel, gbl, gbc, 0, 3, 1, 1, none, west, 0, 0);
  add(pnlPM, p_manage, gbl, gbc, 1, 3, 1, 1, none, west, 0, 0);

  add(pnlPM, dumy1, gbl, gbc, 2, 0, 1, 3, none, center, 0, 0);
  add(pnlPM, pllabel, gbl, gbc, 3, 0, 1, 1, none, west, 0, 0);
  add(pnlPM, plchoice, gbl, gbc, 4, 0, 1, 1, none, west, 0, 0);

  add(pnlPM, asslabel, gbl, gbc, 3, 1, 1, 1, none, west, 0, 0);
  add(pnlPM, asslabel_v, gbl, gbc, 4, 1, 1, 1, none, west, 0, 0);
}


/* Set up a configuration panel for IPC. */

private void setPnlIPC(){
  int none = GridBagConstraints.NONE;
  int hori = GridBagConstraints.HORIZONTAL;
```

```
    int vert = GridBagConstraints.VERTICAL;
    int both = GridBagConstraints.BOTH;
    int center = GridBagConstraints.CENTER;
    int north = GridBagConstraints.NORTH;
    int northeast = GridBagConstraints.NORTHEAST;
    int east = GridBagConstraints.EAST;
    int west = GridBagConstraints.WEST;
    GridBagLayout gbl = new GridBagLayout();
    pnlIPC.setLayout(gbl);
    GridBagConstraints gbc = new GridBagConstraints();

    mcklabel = new Label("Message communication keys");
    cf = new Integer(ConfigCurrent.NUMKEYS);
    de = new Integer(ConfigDefault.NUMKEYS);
    mckchoice = new Choice();
    mckchoice.addItem(de.toString());
    mckchoice.addItem("3");
    mckchoice.addItem("4");
    mckchoice.select(cf.toString());

    s1label = new Label("Semaphore1 numbers:");
    cf = new Integer(ConfigCurrent.NUMSEM1);
    de = new Integer(ConfigDefault.NUMSEM1);
    s1choice = new Choice();
    s1choice.addItem(de.toString());
    s1choice.addItem("2");
    s1choice.addItem("3");
    s1choice.select(cf.toString());

    s2label = new Label("Semaphore2 numbers:");
    cf = new Integer(ConfigCurrent.NUMSEM2);
    de = new Integer(ConfigDefault.NUMSEM2);
    s2choice = new Choice();
    s2choice.addItem("1");
    s2choice.addItem(de.toString());
    s2choice.addItem("3");
    s2choice.select(cf.toString());

    sqlabel = new Label("Semaphore quantum:");
    cf = new Integer(ConfigCurrent.SEMQUANTUM);
    de = new Integer(ConfigDefault.SEMQUANTUM);
    sqchoice = new Choice();
    sqchoice.addItem(de.toString());
    sqchoice.addItem("4");
    sqchoice.addItem("6");
    sqchoice.select(cf.toString());

    add(pnlIPC, s1label, gbl, gbc, 0, 0, 1, 1, none, west, 0, 0);
    add(pnlIPC, s1choice, gbl, gbc, 1, 0, 1, 1, none, west, 0, 0);
    add(pnlIPC, s2label, gbl, gbc, 0, 1, 1, 1, none, west, 0, 0);
    add(pnlIPC, s2choice, gbl, gbc, 1, 1, 1, 1, none, west, 0, 0);

    add(pnlIPC, mcklabel, gbl, gbc, 0, 2, 1, 1, none, west, 0, 0);
    add(pnlIPC, mckchoice, gbl, gbc, 1, 2, 1, 1, none, west, 0, 0);
    add(pnlIPC, sqlabel, gbl, gbc, 0, 3, 1, 1, none, west, 0, 0);
    add(pnlIPC, sqchoice, gbl, gbc, 1, 3, 1, 1, none, west, 0, 0);
  }


  /* Set up a configuration panel for Simulation Mode. */

  private void setPnlSM(){
    Label emptyline3;
    int none = GridBagConstraints.NONE;
    int hori = GridBagConstraints.HORIZONTAL;
    int vert = GridBagConstraints.VERTICAL;
    int both = GridBagConstraints.BOTH;
    int center = GridBagConstraints.CENTER;
    int north = GridBagConstraints.NORTH;
    int northeast = GridBagConstraints.NORTHEAST;
```

```
  int east = GridBagConstraints.EAST;
  int west = GridBagConstraints.WEST;

  GridBagLayout gbl = new GridBagLayout();
  pnlSM.setLayout(gbl);
  GridBagConstraints gbc = new GridBagConstraints();
  smlabel = new Label("Simulation Mode:");
  CheckboxGroup cbg = new CheckboxGroup();
  if( ConfigCurrent.MODE == 0) {
    sm_check1 = new Checkbox("Interactive", cbg, true);
    sm_check2 = new Checkbox("Non_Interactive", cbg, false);
  }
  else {
    sm_check1 = new Checkbox("Interactive", cbg, false);
    sm_check2 = new Checkbox("Non_Interactive", cbg, true);
  }
  ptlabel = new Label("Process Operation Types:");

  if( ConfigCurrent.MEMORY_OPERATION)
    pt_check1 = new Checkbox("Memory", null, true);
  else
    pt_check1 = new Checkbox("Memory", null, false);

  if( ConfigCurrent.MESSAGE_OPERATION)
    pt_check2 = new Checkbox("Message", null, true);
  else
    pt_check2 = new Checkbox("Message", null, false);

  if( ConfigCurrent.SEMAPHORE_OPERATION)
    pt_check3 = new Checkbox("Semaphore", null, true);
  else
    pt_check3 = new Checkbox("Semaphore", null, false);

  if( ConfigCurrent.MODE == 0) {
    pnlSMDis();
  }
  emptyline3 = new Label("");
  add(pnlSM, smlabel, gbl, gbc, 0, 0, 3, 1, none, west, 0, 0);
  add(pnlSM, sm_check1, gbl, gbc, 0, 1, 3, 1, none, center, 0, 0);
  add(pnlSM, sm_check2, gbl, gbc, 3, 1, 3, 1, none, center, 0, 0);
  add(pnlSM, emptyline3, gbl, gbc, 0, 2, 6, 1, none, center, 0, 0);
  add(pnlSM, ptlabel, gbl, gbc, 0, 4, 3, 1, none, west, 0, 0);
  add(pnlSM, pt_check1, gbl, gbc, 0, 5, 2, 1, none, center, 0, 0);
  add(pnlSM, pt_check2, gbl, gbc, 2, 5, 2, 1, none, center, 0, 0);
  add(pnlSM, pt_check3, gbl, gbc, 4, 5, 2, 1, none, center, 0, 0);
}


/* Set up a configuration panel for Output Results. */

private void setPnlOR(){
  Label emptyline1, emptyline2;
  int none = GridBagConstraints.NONE;
  int hori = GridBagConstraints.HORIZONTAL;
  int vert = GridBagConstraints.VERTICAL;
  int both = GridBagConstraints.BOTH;
  int center = GridBagConstraints.CENTER;
  int north = GridBagConstraints.NORTH;
  int northeast = GridBagConstraints.NORTHEAST;
  int east = GridBagConstraints.EAST;
  int west = GridBagConstraints.WEST;

  GridBagLayout gbl = new GridBagLayout();
  pnlOR.setLayout(gbl);
  GridBagConstraints gbc = new GridBagConstraints();
  otflabel = new Label("Output to file:");
  CheckboxGroup cbg = new CheckboxGroup();
  if(! ConfigCurrent.OUTPUT_TO_FILE) {
    otf_check1 = new Checkbox("Yes", cbg, false);
    otf_check2 = new Checkbox("No", cbg, true);
```

```
    }
    else {
      otf_check1 = new Checkbox("Yes", cbg, true);
      otf_check2 = new Checkbox("No", cbg, false);
    }

    fnlabel = new Label("File name:");
    fntext = new TextField(ConfigCurrent.OUTPUT_FILE_NAME, 16);
    fntext.setForeground(Color.black);
    fnbutton = new Button("Browse");

    crlabel = new Label("Contents of output:");

    if( ConfigCurrent.OUTPUT_SYSTEM_INF)
      cr_check1 = new Checkbox("System information", null, true);
    else
      cr_check1 = new Checkbox("System information", null, false);

    if( ConfigCurrent.OUTPUT_RESULT_INF)
      cr_check3 = new Checkbox("Simulation result", null, true);
    else
      cr_check3 = new Checkbox("Simulation result", null, false);
    if( ConfigCurrent.OUTPUT_HISTORY_INF)
      cr_check4 = new Checkbox("Execution history", null, true);
    else
      cr_check4 = new Checkbox("Execution history", null, false);

    if(!ConfigCurrent.OUTPUT_TO_FILE)
      pnlORDis();

    emptyline1 = new Label("");
    emptyline2 = new Label("");

    add(pnlOR, otflabel, gbl, gbc, 0, 0, 4, 1, none, west, 0, 0);
    add(pnlOR, otf_check1, gbl, gbc, 4, 0, 4, 1, none, center, 0, 0);
    add(pnlOR, otf_check2, gbl, gbc, 8, 0, 4, 1, none, center, 0, 0);
    add(pnlOR, emptyline1, gbl, gbc, 0, 1, 12, 2, none, center, 0, 0);
    add(pnlOR, fnlabel, gbl, gbc, 0, 3, 4, 1, none, west, 0, 0);
    add(pnlOR, fntext, gbl, gbc, 4, 3, 4, 1, none, west, 0, 0);
    add(pnlOR, fnbutton, gbl, gbc, 8, 3, 4, 1, none, center, 0, 0);
    add(pnlOR, emptyline2, gbl, gbc, 0, 4, 12, 2, none, center, 0, 0);
    add(pnlOR, crlabel, gbl, gbc, 0, 6, 12, 1, none, west, 0, 0);
    add(pnlOR, cr_check1, gbl, gbc, 0, 7, 4, 1, none, west, 0, 0);
    add(pnlOR, cr_check3, gbl, gbc, 4, 7, 4, 1, none, west, 0, 0);
    add(pnlOR, cr_check4, gbl, gbc, 8, 7, 4, 1, none, west, 0, 0);
  }


  /* Disable partial of the configuration panel for Output Results. */

  private void pnlORDis() {
    fntext.disable();
    fnbutton.disable();
    cr_check1.disable();
    cr_check3.disable();
    cr_check4.disable();
  }

  /* Disable partial of the configuration panel for Simulation Modes. */

  private void pnlSMDis() {
    pt_check1.disable();
    pt_check2.disable();
    pt_check3.disable();
  }


  /* Enable partial of the configuration panel for Output Results. */

  private void pnlOREn() {
```

```
  fntext.enable();
  fnbutton.enable();
  cr_check1.enable();
  cr_check3.enable();
  cr_check4.enable();
 }

 /* Enable partial of the configuration panel for Simulation Modes. */

 private void pnlSMEn() {
  pt_check1.enable();
  pt_check2.enable();
  pt_check3.enable();
 }


 /* Set system configuration to default values. */

 private void setDefault() {
  Integer cf;

  if ( currentpnl.equals(pnlS)) {

    cf = new Integer(ConfigDefault.PAGESIZE);
    pchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.DMADELAY);
    dchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.MAX_HISTORY_ITEMS);
    mhchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.WAIT_TIME);
    wtchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.WAIT_TIME_PORTION);
    wtpchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.WAIT_TIME_MIN);
    wtmchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.TIME_UNIT);
    tuchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.MAX_CPU_USE_BITS);
    mcchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.MAX_MEM_USE_BITS);
    mmuchoice.select(cf.toString());

    // When some depended values change in a configuration panel,
    // all dependent configuration values will change accordingly.
    relatedChange();
  }
  else if ( currentpnl.equals(pnlMM)) {
    cf = new Integer(ConfigDefault.MEMSIZE);
    mchoice.select(cf.toString());

    cf = new Integer(ConfigDefault.TOTAL_MEMORY_ACCESS);
    mmchoice.select(cf.toString());

    if(ConfigDefault.MEMORY_MANAGEMENT == EnumPageReplacement.FIFO)
      m_manage.select("FIFO");
    else
      m_manage.select("SECOND CHANCE");

    // When some depended values change in a configuration panel,
    // all dependent configuration values will change accordingly.
    relatedChange();
  }
```

```
  else if ( currentpnl.equals(pnlPM)) {
   cf = new Integer(ConfigDefault.NUMPAGES);
   pnchoice.select(cf.toString());

   cf = new Integer(ConfigDefault.MAXPROCESS);
   mpchoice.select(cf.toString());

   cf = new Integer(ConfigDefault.QUANTUM);
   qchoice.select(cf.toString());

   cf = new Integer(ConfigDefault.PRIORITY_LEVEL);
   plchoice.select(cf.toString());

   if(ConfigDefault.PROCESS_MANAGEMENT == EnumScheduling.ROUND_ROBIN)
    p_manage.select("ROUND ROBIN");
   else
    p_manage.select("PRIORITY");

   // When some depended values change in a configuration panel,
   // all dependent configuration values will change accordingly.
   relatedChange();
  }
  else if ( currentpnl.equals(pnlIPC)) {

   cf = new Integer(ConfigDefault.NUMKEYS);
   mckchoice.select(cf.toString());

   cf = new Integer(ConfigDefault.NUMSEM1);
   s1choice.select(cf.toString());

   cf = new Integer(ConfigDefault.NUMSEM2);
   s2choice.select(cf.toString());

   cf = new Integer(ConfigDefault.SEMQUANTUM);
   sqchoice.select(cf.toString());

   // When some depended values change in a configuration panel,
   // all dependent configuration values will change accordingly.
   relatedChange();
  }
  else if ( currentpnl.equals(pnlSM)) {
   if (ConfigDefault.MODE == 0) {
    sm_check1.setState(true);
    pnlSMDis();
   }
   else {
    sm_check2.setState(true);
    pnlSMEn();
   pt_check1.setState(ConfigDefault.MEMORY_OPERATION);
   pt_check2.setState(ConfigDefault.MESSAGE_OPERATION);
   pt_check3.setState(ConfigDefault.SEMAPHORE_OPERATION);
   }
  }
  else if ( currentpnl.equals(pnlOR)) {
   if (ConfigDefault.OUTPUT_TO_FILE) {
    otf_check1.setState(true);
    pnlOREn();
    fntext.setText(ConfigDefault.OUTPUT_FILE_NAME);
    cr_check1.setState(ConfigDefault.OUTPUT_SYSTEM_INF);
    cr_check3.setState(ConfigDefault.OUTPUT_RESULT_INF);
    cr_check4.setState(ConfigDefault.OUTPUT_HISTORY_INF);
   }
   else {
    otf_check2.setState(true);
    pnlORDis();
   }
  }
}
```

```
/* Set the system configuration to current system configuration,
 * ignore the newly user-defined value. */

private void setCancel() {
 Integer cf;

 cf = new Integer(ConfigCurrent.PAGESIZE);
 pchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.DMADELAY);
 dchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.MAX_HISTORY_ITEMS);
 mhchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.WAIT_TIME);
 wtchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.WAIT_TIME_PORTION);
 wtpchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.WAIT_TIME_MIN);
 wtmchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.TIME_UNIT);
 tuchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.MAX_CPU_USE_BITS);
 mcchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.MAX_MEM_USE_BITS);
 mmuchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.MEMSIZE);
 mchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.TOTAL_MEMORY_ACCESS);
 mmchoice.select(cf.toString());

 if(ConfigCurrent.MEMORY_MANAGEMENT == EnumPageReplacement.FIFO)
  m_manage.select("FIFO");
 else
  m_manage.select("SECOND CHANCE");

 cf = new Integer(ConfigCurrent.NUMPAGES);
 pnchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.MAXPROCESS);
 mpchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.QUANTUM);
 qchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.PRIORITY_LEVEL);
 plchoice.select(cf.toString());

 if(ConfigCurrent.PROCESS_MANAGEMENT == EnumScheduling.ROUND_ROBIN)
  p_manage.select("ROUND ROBIN");
 else
  p_manage.select("PRIORITY");

 cf = new Integer(ConfigCurrent.NUMKEYS);
 mckchoice.select(cf.toString());

 cf = new Integer(ConfigCurrent.NUMSEM1);
 s1choice.select(cf.toString());

 cf = new Integer(ConfigCurrent.NUMSEM2);
 s2choice.select(cf.toString());
```

```
    cf = new Integer(ConfigCurrent.SEMQUANTUM);
    sqchoice.select(cf.toString());

    if (ConfigCurrent.MODE == 0) {
      sm_check1.setState(true);
      pnlSMDis();
    }
    else {
      sm_check2.setState(true);
      pnlSMEn();
    pt_check1.setState(ConfigCurrent.MEMORY_OPERATION);
    pt_check2.setState(ConfigCurrent.MESSAGE_OPERATION);
    pt_check3.setState(ConfigCurrent.SEMAPHORE_OPERATION);
    }

    if (ConfigCurrent.OUTPUT_TO_FILE) {
      otf_check1.setState(true);
      pnlOREn();
      fntext.setText(ConfigCurrent.OUTPUT_FILE_NAME);
      cr_check1.setState(ConfigCurrent.OUTPUT_SYSTEM_INF);
      cr_check3.setState(ConfigCurrent.OUTPUT_RESULT_INF);
      cr_check4.setState(ConfigCurrent.OUTPUT_HISTORY_INF);
    }
    else {
      otf_check2.setState(true);
      pnlORDis();
    }

    // When depended configuration values change, all dependent configuration
    // values will change accordingly.
    relatedConfig();
  }


  /* When depended configuration values change, all dependent configuration
   * values will change accordingly. */

  private void relatedConfig() {
    int i1, i2, i3, i4;
    Integer In;
    i1 = IOFormat.atoi(mchoice.getSelectedItem());
    i2 = IOFormat.atoi(pchoice.getSelectedItem());
    i3 = IOFormat.atoi(pnchoice.getSelectedItem());
    i4 = IOFormat.atoi(mpchoice.getSelectedItem());

    In = new Integer(i4*i3*i2);
    bslabel_v.setText(In.toString());

    In = new Integer(i4*i3);
    pelabel_v.setText(In.toString());

    In = new Integer(i1/i2);
    tmflabel_v.setText(In.toString());

    In = new Integer(i1/i2/4);
    tfflabel_v.setText(In.toString());

    In = new Integer(i1/i2/2);
    rfnlabel_v.setText(In.toString());

    In = new Integer(i2*i3);
    asslabel_v.setText(In.toString());
  }


  /* When some depended values change in a configuration panel,
   * all dependent configuration values will change accordingly. */

  private void relatedChange() {
    int i1, i2, i3, i4;
```

```
 Integer In;
 i1 = IOFormat.atoi(mchoice.getSelectedItem());
 i2 = IOFormat.atoi(pchoice.getSelectedItem());
 i3 = IOFormat.atoi(pnchoice.getSelectedItem());
 i4 = IOFormat.atoi(mpchoice.getSelectedItem());

 if ( currentpnl.equals(pnlS)) {
   In = new Integer(i4*i3*i2);
   bslabel_v.setText(In.toString());

   In = new Integer(i4*i3);
   pelabel_v.setText(In.toString());
 }
 else if (currentpnl.equals(pnlMM)) {
   In = new Integer(i1/i2);
   tmflabel_v.setText(In.toString());

   In = new Integer(i1/i2/4);
   tfflabel_v.setText(In.toString());

   In = new Integer(i1/i2/2);
   rfnlabel_v.setText(In.toString());
 }
 else if (currentpnl.equals(pnlPM)) {
   In = new Integer(i2*i3);
   asslabel_v.setText(In.toString());
 }
}


/* Set the partial current system configuration by the vaules on
 * configuration window. */

private void setConfig() {
 String s ;
 ConfigCurrent.MEMSIZE = IOFormat.atoi(mchoice.getSelectedItem());
 ConfigCurrent.PAGESIZE = IOFormat.atoi(pchoice.getSelectedItem());
 ConfigCurrent.NUMPAGES  = IOFormat.atoi(pnchoice.getSelectedItem());
 ConfigCurrent.MAXPROCESS = IOFormat.atoi(mpchoice.getSelectedItem());
 ConfigCurrent.QUANTUM = IOFormat.atoi(qchoice.getSelectedItem());
 ConfigCurrent.TOTAL_MEMORY_ACCESS = IOFormat.atoi(mmchoice.getSelectedItem());
 ConfigCurrent.PRIORITY_LEVEL = IOFormat.atoi(plchoice.getSelectedItem());
 ConfigCurrent.DMADELAY = IOFormat.atoi(dchoice.getSelectedItem());
 ConfigCurrent.NUMKEYS = IOFormat.atoi(mckchoice.getSelectedItem());
 ConfigCurrent.NUMSEM1 = IOFormat.atoi(s1choice.getSelectedItem());
 ConfigCurrent.NUMSEM2 = IOFormat.atoi(s2choice.getSelectedItem());
 ConfigCurrent.SEMQUANTUM = IOFormat.atoi(sqchoice.getSelectedItem());
 ConfigCurrent.MAX_HISTORY_ITEMS = IOFormat.atoi(mhchoice.getSelectedItem());
 ConfigCurrent.WAIT_TIME = IOFormat.atoi(wtchoice.getSelectedItem());
 ConfigCurrent.WAIT_TIME_PORTION = IOFormat.atoi(wtpchoice.getSelectedItem());
 ConfigCurrent.WAIT_TIME_MIN = IOFormat.atoi(wtmchoice.getSelectedItem());
 ConfigCurrent.ADDRSPACESIZE = IOFormat.atoi(asslabel.getText());
 ConfigCurrent.MEMORYFRAMES = IOFormat.atoi(tmflabel.getText());
 ConfigCurrent.BACKSTORESIZE = IOFormat.atoi(bslabel.getText());
 ConfigCurrent.PAGEENTRY = IOFormat.atoi(pelabel.getText());
 ConfigCurrent.FREE_FRAME_THRASHOLD = IOFormat.atoi(tfflabel.getText());
 ConfigCurrent.RETRIEVED_FRAMES = IOFormat.atoi(rfnlabel.getText());

 s = p_manage.getSelectedItem();
 if (s.compareTo("ROUND ROBIN") == 0)
   ConfigCurrent.PROCESS_MANAGEMENT = EnumScheduling.ROUND_ROBIN;
 else if (s.compareTo("PRIORITY") == 0)
   ConfigCurrent.PROCESS_MANAGEMENT = EnumScheduling.PRIORITY;

 s = m_manage.getSelectedItem();
 if (s.compareTo("FIFO") == 0)
   ConfigCurrent.MEMORY_MANAGEMENT = EnumPageReplacement.FIFO;
 else if (s.compareTo("SECOND CHANCE") == 0)
   ConfigCurrent.MEMORY_MANAGEMENT = EnumPageReplacement.SECOND_CHANCE;
```

```
    ConfigCurrent.TIME_UNIT = IOFormat.atoi(tuchoice.getSelectedItem());
    ConfigCurrent.MAX_CPU_USE_BITS = IOFormat.atoi(mcchoice.getSelectedItem());
    ConfigCurrent.MAX_MEM_USE_BITS = IOFormat.atoi(mmuchoice.getSelectedItem());

    if (sm_check1.getState())
      ConfigCurrent.MODE = 0;
    else
      ConfigCurrent.MODE = 1;
    ConfigCurrent.MEMORY_OPERATION = pt_check1.getState();
    ConfigCurrent.MESSAGE_OPERATION = pt_check2.getState();
    ConfigCurrent.SEMAPHORE_OPERATION = pt_check3.getState();
    ConfigCurrent.OUTPUT_TO_FILE = otf_check1.getState();
    ConfigCurrent.OUTPUT_FILE_NAME = fntext.getText();
    ConfigCurrent.OUTPUT_SYSTEM_INF = cr_check1.getState();
    ConfigCurrent.OUTPUT_RESULT_INF = cr_check3.getState();
    ConfigCurrent.OUTPUT_HISTORY_INF = cr_check4.getState();

    // Store the current system configuration values to Config.current file.
    writeData();
  }


  /* Print current system configuration to standard output. */

  private void printConfig() {
    System.out.println("NREGS = "+ConfigCurrent.NREGS);
    System.out.println("MEMSIZE = "+ConfigCurrent.MEMSIZE);
    System.out.println("PAGESIZE = "+ConfigCurrent.PAGESIZE);
    System.out.println("NUMPAGES = "+ConfigCurrent.NUMPAGES);
    System.out.println("ADDRSPACESIZE = "+ConfigCurrent.ADDRSPACESIZE);
    System.out.println("MEMORYFRAMES = "+ConfigCurrent.MEMORYFRAMES);
    System.out.println("MAXPROCESS = "+ConfigCurrent.MAXPROCESS);
    System.out.println("BACKSTORESIZE = "+ConfigCurrent.BACKSTORESIZE);
    System.out.println("PAGEENTRY = "+ConfigCurrent.PAGEENTRY);
    System.out.println("QUANTUM = "+ConfigCurrent.QUANTUM);
    System.out.println("TOTAL_MEMORY_ACCESS = "+ConfigCurrent.TOTAL_MEMORY_ACCESS);
    System.out.println("FREE_FRAME_THRASHOLD = "+ConfigCurrent.FREE_FRAME_THRASHOLD);
    System.out.println("RETRIEVED_FRAMES = "+ConfigCurrent.RETRIEVED_FRAMES);
    System.out.println("PROCESS_MANAGEMENT = "+ConfigCurrent.PROCESS_MANAGEMENT);
    System.out.println("MEMORY_MANAGEMENT = "+ConfigCurrent.MEMORY_MANAGEMENT);
    System.out.println("PRIORITY_LEVEL = "+ConfigCurrent.PRIORITY_LEVEL);
    System.out.println("DMADELAY = "+ConfigCurrent.DMADELAY);
    System.out.println("NUMKEYS = "+ConfigCurrent.NUMKEYS);
    System.out.println("NUMSEM1 = "+ConfigCurrent.NUMSEM1);
    System.out.println("NUMSEM2 = "+ConfigCurrent.NUMSEM2);
    System.out.println("SEMQUANTUM = "+ConfigCurrent.SEMQUANTUM);
    System.out.println("MAX_HISTORY_ITEMS = "+ConfigCurrent.MAX_HISTORY_ITEMS);
    System.out.println("WAIT_TIME = "+ConfigCurrent.WAIT_TIME);
    System.out.println("WAIT_TIME_PORTION = "+ConfigCurrent.WAIT_TIME_PORTION);
    System.out.println("WAIT_TIME_MIN = "+ConfigCurrent.WAIT_TIME_MIN);
    System.out.println("TIME_UNIT = "+ConfigCurrent.TIME_UNIT);

    System.out.println("MAX_CPU_USE_BITS = "+ConfigCurrent.MAX_CPU_USE_BITS);
    System.out.println("MAX_MEM_USE_BITS = "+ConfigCurrent.MAX_MEM_USE_BITS);

    System.out.println("MODE = "+ConfigCurrent.MODE);
    System.out.println("MEMORY_OPERATION = "+ConfigCurrent.MEMORY_OPERATION);
    System.out.println("MESSAGE_OPERATION = "+ConfigCurrent.MESSAGE_OPERATION);
    System.out.println("SEMAPHORE_OPERATION = "+ConfigCurrent.SEMAPHORE_OPERATION);

    System.out.println("OUTPUT_TO_FILE = "+ConfigCurrent.OUTPUT_TO_FILE);
    System.out.println("OUTPUT_FILE_NAME = "+ConfigCurrent.OUTPUT_FILE_NAME);

    System.out.println("OUTPUT_SYSTEM_INF = "+ConfigCurrent.OUTPUT_SYSTEM_INF);
    System.out.println("OUTPUT_RESULT_INF = "+ConfigCurrent.OUTPUT_RESULT_INF);
    System.out.println("OUTPUT_HISTORY_INF = "+ConfigCurrent.OUTPUT_HISTORY_INF);
  }


  /* Set a current system configuration by name. */
```

```
private void getConfigData(String name, int value) {
 if(name.compareTo("MEMSIZE") == 0)
  ConfigCurrent.MEMSIZE = value;
 if(name.compareTo("PAGESIZE") == 0)
  ConfigCurrent.PAGESIZE = value;
 if(name.compareTo("NUMPAGES") == 0)
  ConfigCurrent.NUMPAGES = value;
 if(name.compareTo("MAXPROCESS") == 0)
  ConfigCurrent.MAXPROCESS = value;
 if(name.compareTo("QUANTUM") == 0)
  ConfigCurrent.QUANTUM = value;
 if(name.compareTo("TOTAL_MEMORY_ACCESS") == 0)
  ConfigCurrent.TOTAL_MEMORY_ACCESS = value;
 if(name.compareTo("PROCESS_MANAGEMENT") == 0)
  ConfigCurrent.PROCESS_MANAGEMENT = value;
 if(name.compareTo("MEMORY_MANAGEMENT") == 0)
  ConfigCurrent.MEMORY_MANAGEMENT = value;
 if(name.compareTo("PRIORITY_LEVEL") == 0)
  ConfigCurrent.PRIORITY_LEVEL = value;
 if(name.compareTo("DMADELAY") == 0)
  ConfigCurrent.DMADELAY = value;
 if(name.compareTo("NUMKEYS") == 0)
  ConfigCurrent.NUMKEYS = value;
 if(name.compareTo("NUMSEM1") == 0)
  ConfigCurrent.NUMSEM1 = value;
 if(name.compareTo("NUMSEM2") == 0)
  ConfigCurrent.NUMSEM2 = value;
 if(name.compareTo("SEMQUANTUM") == 0)
  ConfigCurrent.SEMQUANTUM = value;
 if(name.compareTo("MAX_HISTORY_ITEMS") == 0)
  ConfigCurrent.MAX_HISTORY_ITEMS = value;
 if(name.compareTo("WAIT_TIME") == 0)
  ConfigCurrent.WAIT_TIME = value;
 if(name.compareTo("WAIT_TIME_PORTION") == 0)
  ConfigCurrent.WAIT_TIME_PORTION = value;
 if(name.compareTo("WAIT_TIME_MIN") == 0)
  ConfigCurrent.WAIT_TIME_MIN = value;
 if(name.compareTo("TIME_UNIT") == 0)
  ConfigCurrent.TIME_UNIT = value;
 if(name.compareTo("MAX_CPU_USE_BITS") == 0)
  ConfigCurrent.MAX_CPU_USE_BITS = value;
 if(name.compareTo("MAX_MEM_USE_BITS") == 0)
  ConfigCurrent.MAX_MEM_USE_BITS = value;
 if(name.compareTo("MODE") == 0)
  ConfigCurrent.MODE = value;
 if(name.compareTo("MEMORY_OPERATION") == 0) {
  if(value == 1)
   ConfigCurrent.MEMORY_OPERATION = true;
  else
   ConfigCurrent.MEMORY_OPERATION = false;
 }
 if(name.compareTo("MESSAGE_OPERATION") == 0) {
  if(value == 1)
   ConfigCurrent.MESSAGE_OPERATION = true;
  else
   ConfigCurrent.MESSAGE_OPERATION = false;
 }
 if(name.compareTo("SEMAPHORE_OPERATION") == 0) {
  if(value == 1)
   ConfigCurrent.SEMAPHORE_OPERATION = true;
  else
   ConfigCurrent.SEMAPHORE_OPERATION = false;
 }
 if(name.compareTo("OUTPUT_TO_FILE") == 0)
  if(value == 1)
   ConfigCurrent.OUTPUT_TO_FILE = true;
  else
   ConfigCurrent.OUTPUT_TO_FILE = false;
```

```
 if(name.compareTo("OUTPUT_SYSTEM_INF") == 0) {
  if(value == 1)
   ConfigCurrent.OUTPUT_SYSTEM_INF = true;
  else
   ConfigCurrent.OUTPUT_SYSTEM_INF = false;
 }
 if(name.compareTo("OUTPUT_RESULT_INF") == 0) {
  if(value == 1)
   ConfigCurrent.OUTPUT_RESULT_INF = true;
  else
   ConfigCurrent.OUTPUT_RESULT_INF = false;
 }
 if(name.compareTo("OUTPUT_HISTORY_INF") == 0) {
  if(value == 1)
   ConfigCurrent.OUTPUT_HISTORY_INF = true;
  else
   ConfigCurrent.OUTPUT_HISTORY_INF = false;
 }
}


/* Calculate all dependent current configuration values according to their
 * depended values. */

private void calculateData() {
 ConfigCurrent.ADDRSPACESIZE = ConfigCurrent.PAGESIZE * ConfigCurrent.NUMPAGES;
 ConfigCurrent.MEMORYFRAMES = ConfigCurrent.MEMSIZE / ConfigCurrent.PAGESIZE;
 ConfigCurrent.BACKSTORESIZE = ConfigCurrent.MAXPROCESS * ConfigCurrent.NUMPAGES * ConfigCurrent.PAGESIZE;
 ConfigCurrent.PAGEENTRY = ConfigCurrent.MAXPROCESS * ConfigCurrent.NUMPAGES;
 ConfigCurrent.FREE_FRAME_THRASHOLD = (ConfigCurrent.MEMSIZE / ConfigCurrent.PAGESIZE)/4;
 ConfigCurrent.RETRIEVED_FRAMES = (ConfigCurrent.MEMSIZE / ConfigCurrent.PAGESIZE)/2;
}


/* Gnerate the current system configuration from Config.current file */

private void readData(){
 try {
  String s;
  StringTokenizer t;
  String name;
  int value;
  DataInputStream is = new DataInputStream (
                new FileInputStream("Config.current"));

  // Get the number of configuration values in Config.current.
  int n = IOFormat.atoi(is.readLine());

  for (int i = 0; i< n; i++) {
   s = is.readLine();
   t = new StringTokenizer(s, "|");
   name = t.nextToken();
   if(name.compareTo("OUTPUT_FILE_NAME") == 0)
    ConfigCurrent.OUTPUT_FILE_NAME = t.nextToken();
   else {
    value = IOFormat.atoi(t.nextToken());
    getConfigData(name, value); // Set a current system configuration by name.
   }
   // Calculate all dependent current configuration values
   // according to their depended values.
   calculateData();
  }
  is.close();
 }
 catch(IOException e) {
  System.out.print("Error: " + e);
  System.exit(1);
 }
}
```

```
/* Store the current system configuration values to Config.current file. */

public void writeData(){
 try {
  PrintStream os = new PrintStream(new FileOutputStream("Config.current"));
  IOFormat.print(os, "%d\n", 30);
  output(os, "MEMSIZE", ConfigCurrent.MEMSIZE);
  output(os, "PAGESIZE", ConfigCurrent.PAGESIZE);
  output(os, "NUMPAGES", ConfigCurrent.NUMPAGES);
  output(os, "MAXPROCESS", ConfigCurrent.MAXPROCESS);
  output(os, "QUANTUM", ConfigCurrent.QUANTUM);
  output(os, "TOTAL_MEMORY_ACCESS", ConfigCurrent.TOTAL_MEMORY_ACCESS);
  output(os, "PROCESS_MANAGEMENT", ConfigCurrent.PROCESS_MANAGEMENT);
  output(os, "MEMORY_MANAGEMENT", ConfigCurrent.MEMORY_MANAGEMENT );
  output(os, "PRIORITY_LEVEL", ConfigCurrent.PRIORITY_LEVEL);
  output(os, "DMADELAY", ConfigCurrent.DMADELAY);
  output(os, "NUMKEYS", ConfigCurrent.NUMKEYS);
  output(os, "NUMSEM1", ConfigCurrent.NUMSEM1);
  output(os, "NUMSEM2", ConfigCurrent.NUMSEM2);
  output(os, "SEMQUANTUM", ConfigCurrent.SEMQUANTUM);
  output(os, "MAX_HISTORY_ITEMS", ConfigCurrent.MAX_HISTORY_ITEMS);
  output(os, "WAIT_TIME", ConfigCurrent.WAIT_TIME);
  output(os, "WAIT_TIME_PORTION", ConfigCurrent.WAIT_TIME_PORTION);
  output(os, "WAIT_TIME_MIN", ConfigCurrent.WAIT_TIME_MIN);
  output(os, "TIME_UNIT", ConfigCurrent.TIME_UNIT);
  output(os, "MAX_CPU_USE_BITS", ConfigCurrent.MAX_CPU_USE_BITS);
  output(os, "MAX_MEM_USE_BITS", ConfigCurrent.MAX_MEM_USE_BITS);
  output(os, "MODE", ConfigCurrent.MODE);
  output(os, "MEMORY_OPERATION", ConfigCurrent.MEMORY_OPERATION);
  output(os, "MESSAGE_OPERATION", ConfigCurrent.MESSAGE_OPERATION);
  output(os, "SEMAPHORE_OPERATION", ConfigCurrent.SEMAPHORE_OPERATION);
  output(os, "OUTPUT_TO_FILE", ConfigCurrent.OUTPUT_TO_FILE);
  output(os, "OUTPUT_FILE_NAME", ConfigCurrent.OUTPUT_FILE_NAME);
  output(os, "OUTPUT_SYSTEM_INF", ConfigCurrent.OUTPUT_SYSTEM_INF);
  output(os, "OUTPUT_RESULT_INF", ConfigCurrent.OUTPUT_RESULT_INF);
  output(os, "OUTPUT_HISTORY_INF", ConfigCurrent.OUTPUT_HISTORY_INF);
  os.close();
 }
 catch(IOException e) {
  System.out.print("Error: " + e);
  System.exit(1);
 }
}


/* Use "name\tvalue" format to store configuration values in file. */

private void output(PrintStream os, String name, int value){
 IOFormat.print(os, "%s\t", name);
 IOFormat.print(os, "%d\n", value);
}


/* Use "name\tboolean value" format to store configuration values in file. */

private void output(PrintStream os, String name, boolean b){
 IOFormat.print(os, "%s\t", name);
 if ( b )
  IOFormat.print(os, "%d\n", 1);
 else
  IOFormat.print(os, "%d\n", 0);
}


/* Use "name\tstring" format to store configuration values in file. */

private void output(PrintStream os, String name, String str){
 IOFormat.print(os, "%s\t", name);
 IOFormat.print(os, "%s\n", str);
```

```
    }

    /* Use GridBagLayout to arrange a component in a configuration panel. */

    private void add(Panel pnl, Component c, GridBagLayout gbl,
        GridBagConstraints gbc,
        int x, int y, int w, int h, int f, int a, int xw, int yw) {
      gbc.gridx = x;
      gbc.gridy = y;
      gbc.gridwidth = w;
      gbc.gridheight = h;

      gbc.fill = f;
      gbc.weightx = xw;
      gbc.weighty = yw;
      gbc.anchor = a;

      gbl.setConstraints(c, gbc);
      pnl.add(c);
    }


}
```

A.5. DMARequest.java

```
/* This class models a DMA request. */

public class DMARequest{

 private int memFrame; // Memory frame number.
 private int bsFrame;  // Backstore frame number.
 private int direction; // Direction of DMA.


 /* The constructor initializes a DMA request. */

 public DMARequest(int memFrame, int bsFrame, int direction){
  this.memFrame = memFrame;
  this.bsFrame = bsFrame;
  this.direction = direction;
 }


 /* Return memory frame number */

 public int getMemFrame(){
  return memFrame;
 }


 /* Return backstore frame number */

 public int getBsFrame(){
  return bsFrame;
 }


 /* Return DMA direction. */

 public int getDirection(){
  return direction;
 }


}
```

A.6. EnumDMATransferDirection.java

```
/* System constant -- DMA transfer direction. */

public class EnumDMATransferDirection{

  public static final int MEMTOBS = 1; // Memory to backstore.
  public static final int BSTOMEM = 2; // Backstore to memory.


}
```

A.7. EnumInterrupt.java

```
/* System constant -- interrupt types. */

public class EnumInterrupt{

 public static final int CLOCK = 1;        // Clock interrupt.
 public static final int DMA = 2;          // DMA interrupt.
 public static final int PAGEFAULT = 3;    // Page fault interrupt.
 public static final int ILLEGALADDRESS = 4; // Illegal address interrupt.
 public static final int SOFTWARE = 5;     // Software interrupt.
 public static final int RESOURCEDONE = 6;  // Resource usage done interrupt.


}
```

A.8. EnumPageReplacement.java

```
/* System constant -- choice of page replacement algorithms. */

public class EnumPageReplacement{

  public static final int FIFO = 0;        // FIFO algorithm.
  public static final int SECOND_CHANCE = 1; // Second-chance algorithm.


}
```

A.9. EnumProcessState.java

```
/* System constant -- process states. */

public class EnumProcessState{

  public static final int Running = 1;      // Running state.
  public static final int Ready = 2;        // Ready to run state.
  public static final int Terminated = 3;   // Terminated state.
  public static final int Blocked = 4;      // Blocked state.
  public static final int MessageWait = 5;  // Waiting for message state.
  public static final int SemaphoreWait = 6; // Waiting for semaphore state.


}
```

A.10. EnumScheduling.java

```
/* System constant -- choice of scheduling algorithm. */

public class EnumScheduling{

  public static final int ROUND_ROBIN = 0; // Round-robin algorithm.
  public static final int PRIORITY = 1;    // Priority algorithm.


}
```

A.11. EnumSimulationMode.java

```
/* System constant -- choice of simulation mode. */

public class EnumSimulationMode{

  public static final int INTERACTIVE = 0;     // Interactive mode.
  public static final int NON_INTERACTIVE = 1; // Non-interactive mode.


}
```

A.12. EnumSystemCallNum.java

```
/* System constant -- system call numbers. */

public class EnumSystemCallNum{

  public static final int ERROR = 0;  // System error.
  public static final int FORK = 1;   // Forking a new process.
  public static final int GETPID = 2; // Getting process ID.
  public static final int YIELD = 3;  // Yielding execution.
  public static final int MSGSND = 4; // Sending a message.
  public static final int MSGRCV = 5; // Receiving a message.
  public static final int SemPop = 6; // Applying a semaphore.
  public static final int SemVop = 7; // Releasing a semaphore.


}
```

A.13. Executer.java

```
import java.util.*;


/* This class is responsible for executing a simulated user process or
 * an idle process. */

public class Executer{

 private Machine     machine;      // Machine.
 private ProcessMgr  processmgr;   // Process manager.
 private ResourceMgr resourcemgr;  // Resource manager.
 private SystemCall  systemcall;   // System call.
 private History     history;      // Execution history.
 private boolean     debugflag;    // Debug flag.


 /* The constructor initializes Executer. */

 public Executer(Machine machine, History history){
  this.machine = machine;
  this.history = history;
 }


 /* Set process manager. */

 public void setProcessMgr(ProcessMgr processmgr){
  this.processmgr = processmgr;
 }


 /* Set resource manager. */

 public void setResourceMgr(ResourceMgr resourcemgr){
  this.resourcemgr = resourcemgr;
 }


 /* Set system call. */

 public void setSystemCall(SystemCall systemcall){
  this.systemcall = systemcall;
 }


 /* Simulated execution of idle process. */

 public void idleOp(){
  debug("Idle");
  machine.idle();
 }


 /* Simulated execution of a user process. */

 public void executeProcess(PCB pcb, int maxproc){
  debug("Execute");
  int pid;
  pid = pcb.getID(); // Get process ID.
  UserProcess userpro = pcb.getUserPro();
  Vector orderOp = userpro.getOrderOp(); // Get user process operation
                                         // order vector.

  if(pid == maxproc)  // By default, the idle process has the maximum
                      // process number.
   idleOp();
```

```
  else {
   while (!orderOp.isEmpty()) {
    String op = (String) orderOp.elementAt(0);
    if (op.compareTo("Mem") == 0)
     memoryOp(pcb);              // Memory access operation.
    if (op.compareTo("Msg") == 0)
     messageOp(pcb);             // Message passing operation.
    if (op.compareTo("Sem") == 0)
     semaphoreOp(pcb);           // Semaphore operation.
   }
   debug("Process"+pcb.getID()+" is exiting");
   debug("Process"+processmgr.getCurrPro().getID()+" is current process");

   // At this point, the execution is finished.
   processmgr.exit();
  }
 }


/* Turn debugging on or off. */

public void setDebug (boolean flag) {
 debugflag = flag;
}


/* Simulated execution of memory access operation. */

private void memoryOp(PCB proc){
 debug("Memory");

 UserProcess pro = proc.getUserPro();
 int addr;      // Virtual address.
 int reg = 1;   // Register number.
 PCB curproc = processmgr.getCurrPro();

 history.writeHistory(curproc, "is manipulating memories");

 while(!pro.memoryAccessDone()){
  addr = pro.memAddr(); // Get the address.
  curproc.decQuantum(); // Decrease quantum by one.
  machine.memory2Register(addr, reg); // Load memory frame to register.
  pro.accSucc();       // Advance the address to the next one.

  // memory2Register is considered an atomic operation, so
  // time is advanced after the operation.
  machine.tick();
 }

 pro.removeOp("Mem"); // Remove memory operation from the user process.
}


/* Simulated execution of message passing operation. */

private void messageOp(PCB proc){
 debug("Message");

 UserProcess pro = proc.getUserPro();

 // Message sending operation is never blocked.
 if (pro.getSendMsgOp()){
  history.writeHistory(proc, "is sending out messages");
  systemcall.msgSnd(pro.getMsgSendKey(), proc.getID(), 0, null);
 }

 // Message receiving operation may be blocked, if the message
 // wanted is not there in the message queue yet.
 if (pro.getReceiveMsgOp()){
  history.writeHistory(proc, "is about to receive messages");
```

```
      systemcall.msgRcv(pro.getMsgReceiveKey());
    }

    // When both sending and receiving operations are done,
    // the message passing operation is removed from the user process.
    if (!pro.getSendMsgOp() && !pro.getReceiveMsgOp())
      pro.removeOp("Msg");
  }


  /* Simulated execution of semaphore operation */

  private void semaphoreOp(PCB proc){
    debug("Semaphore");

    UserProcess pro = proc.getUserPro();

    // Apply for the resource, if it is not obtained yet.
    if (!pro.getResourceObtained()){
      history.writeHistory(proc, "is about to request for semaphores");
      systemcall.semPop();
    }

    if (pro.getResourceObtained())

      // The user process is not done using the resource.
      if (!pro.getResourceDone()){
        history.writeHistory(proc, "is about to manipulate the resource(s)");
        resourcemgr.manipulateRes(proc);
      }
      else { // Release the resource.
        pro.removeOp("Sem");
        history.writeHistory(proc, "is releasing the semaphores");
        systemcall.semVop();
      }
  }


  /* Print out the debugging information */

  private void debug (String str) {
    if (debugflag)
      System.out.println( "Executer: " + str );
  }


}
```

A.14. Help.java

```
import java.io.*;
import java.awt.*;


/* This class is to pop up a window for providing explanations about a variety
 * of questions the user may raise when using the system. */

public class Help extends Frame {
 private MenuBar menuBar;              // Contains menus.
 private Menu usageMenu;               // Usage menu.
 private Menu designMenu;              // Design menu.
 private Menu interfaceMenu;           // Interface menu.
 private Menu aboutMenu;               // About menu.
 private MenuItem UOSMenuItem;         // Underling operating system
                                       // menu item.
 private MenuItem configrutionMenuItem;  // Configuration menu item.
 private MenuItem executionMenuItem;     // Execution menu item.
 private MenuItem emMenuItem;          // Execution Mode menu item.
 private MenuItem orMenuItem;          // Output Results menu item.
 private MenuItem msMenuItem;          // Monitoring Status menu item.
 private MenuItem mpMenuItem;          // Monitoring Performance menu item.
 private MenuItem authorMenuItem;      // Author menu item.
 private MenuItem versionMenuItem;     // Version menu item.
 private Panel btnBar;                 // Button panel.
 private Button OKButton;              // OK button.
 private TextArea textArea;            // Text Area.


 /* Constructor for Help class */

 public Help() {
  setTitle("Help");

  menuBar = new MenuBar();
  usageMenu = new Menu("Usage");
  usageMenu.add( configrutionMenuItem = new MenuItem("Configuration"));
  usageMenu.addSeparator();
  usageMenu.add( executionMenuItem = new MenuItem("Execution"));
  usageMenu.add( emMenuItem = new MenuItem("Execution Mode"));
  usageMenu.addSeparator();
  usageMenu.add( orMenuItem = new MenuItem("Output Results"));
  menuBar.add(usageMenu);

  designMenu = new Menu("Design");
  interfaceMenu = new Menu("Interface");
  interfaceMenu.add(msMenuItem = new MenuItem("Monitoring Status"));
  interfaceMenu.add(mpMenuItem = new MenuItem("Monitoring Performance"));
  designMenu.add(interfaceMenu);
  designMenu.addSeparator();

  UOSMenuItem = new MenuItem("Underlying Operating System");
  designMenu.add(UOSMenuItem);

  menuBar.add(designMenu);

  aboutMenu = new Menu("About");
  aboutMenu.add( authorMenuItem = new MenuItem("Author"));
  aboutMenu.add(versionMenuItem = new MenuItem("Version"));
  menuBar.add(aboutMenu);

  setLayout( new BorderLayout() );
  setMenuBar(menuBar);

  textArea = new TextArea(30, 70);
  textArea.setEditable(false);
  add("Center", textArea);
```

```
    btnBar = new Panel();
    OKButton = new Button("OK");
    btnBar.add(OKButton);

    add("South", btnBar);
   }


   /* Help window event-driven action */

   public boolean action(Event evt, Object arg) {
    if (evt.target instanceof Button) {
     if (arg.equals("OK"))
      dispose();
    }
    else if (evt.target instanceof MenuItem) {
     if(arg.equals("Configuration"))
      displayHelp("Configuration.txt");
     else if(arg.equals("Execution"))
      displayHelp("Execution.txt");
     else if(arg.equals("Execution Mode"))
      displayHelp("ExecutionMode.txt");
     else if(arg.equals("Output Results"))
      displayHelp("OutputResults.txt");
     else if(arg.equals("Monitoring Status"))
      displayHelp("MonitoringStatus.txt");
     else if(arg.equals("Underlying Operating System"))
      displayHelp("OperatingSystem.txt");
     else if(arg.equals("Monitoring Performance"))
      displayHelp("MonitoringPerformance.txt");
     else if(arg.equals("Author"))
      displayHelp("Author.txt");
     else if(arg.equals("Version"))
      displayHelp("Version.txt");
    }
    else
     return false;
    return true;
   }


   /* Display corresponding contents of selected menu item. */

   private void displayHelp(String s) {
    File file = new File(s);
    try {
     FileInputStream fis = new FileInputStream ( file );
     byte [] data = new byte [ fis.available() ];
     fis.read( data );
     String disdata = new String(data, 8);
     textArea.setText(disdata);
    }
    catch ( IOException e ) {
    }
   }


   /* Handle events */

   public boolean handleEvent(Event evt) {
    if (evt.id == Event.WINDOW_DESTROY && evt.target == this)
     System.exit(0);
    return super.handleEvent(evt);
   }


}
```

A.15. History.java

```
import java.util.*;
import java.awt.*;
import java.lang.*;
import java.io.*;


/* This class is responsible for displaying execution history on the user
 * interface and printing it to a file. */

public class History{

 private List historyList;    // History list on the interface.
 private Vector historyVector; // History vector containing items.
 private boolean bvisual;     // Indicates if items are visible on interface.
 private boolean frozen;      // Indicates if history displaying if frozen.
 private Yield yield;         // Object to sleep for a while.


 /* The constructor initializes History. */

 public History(Yield yield){
  this.yield = yield;
  historyVector = new Vector();
 }


 /* Set the display part on the user interface. */

 public void setDisplay(List historyList, boolean bvisual){
  this.historyList = historyList;
  this.bvisual = bvisual;
  frozen = false;
 }


 /* Write the execution history item onto the history list (user interface)
  * and into the history vector (for printing out into a file). */

 public void writeHistory(PCB proc, String msg){

  // Sleeps if the execution is frozen.
  while (frozen)
   try{
    yield.sleep(1000);
   } catch (InterruptedException e) { };

  String Message; // Message to be displayed.
  String timeMsg; // Time when the message is issued.

  int id = proc.getID();
  if (id != ConfigCurrent.MAXPROCESS)
   Message = "Process " + proc.getID() + " - " + msg;
  else
   Message = "Null Process " + " - " + msg;

  if(bvisual){
   Date   theDateTime = new Date();
   Long   lDateTime;
   lDateTime = new Long(theDateTime.getTime());
   timeMsg = lDateTime.toString();
   timeMsg = timeMsg + " - " + Message;
   historyList.addItem(timeMsg,0);   // Add the item onto the interface.
   historyVector.addElement(timeMsg); // Add the item into the vector.

   // When the number of items in both history list and vector exceeds
   // a maximum number, the oldest one is removed.
   if(historyList.countItems() > ConfigCurrent.MAX_HISTORY_ITEMS){
```

```
      historyList.delItem(ConfigCurrent.MAX_HISTORY_ITEMS);
      historyVector.removeElementAt(0);
     }
    }
    else // Otherwise, history item is displayed on standard output.
     System.out.println(Message);
  }


  /* Freeze the displaying of history items. */

  public void freezeExec(){

   // Allow some time to freeze Machine completely first.
   try{
    yield.sleep(100);
   } catch (InterruptedException e) { };

   frozen = true;
  }


  /* Un-freeze the displaying of history items. */

  public void unFreeze(){
   frozen = false;
  }


  /* Print out history items to a file */

  public void printout(PrintStream os) throws IOException{

   String historyItem;

   IOFormat.print(os, "%s\n", "************************************************");
   IOFormat.print(os, "%s\n", "*       EXECUTION HISTORY INFORMATION       *");
   IOFormat.print(os, "%s\n", "************************************************");

   for (Enumeration e = historyVector.elements() ; e.hasMoreElements() ;) {
    historyItem = (String) e.nextElement();
    IOFormat.print(os, "%s\n", historyItem);
   }

   IOFormat.print(os, "%s\n", " ");

   // Outputs to a file occurs when the simulation is finished, so all items
   // are removed from the vector after the printout.
   historyVector.removeAllElements();
  }


}
```

A.16. IOFormat.java

```
/* This class is a modified version of Format class from Core Java book. */

/*
 * Gary Cornell and Cay S. Horstmann, Core Java (Book/CD-ROM)
 * Published By SunSoft Press/Prentice-Hall
 * Copyright (C) 1996 Sun Microsystems Inc.
 * All Rights Reserved. ISBN 0-13-596891-7
 *

 * Permission to use, copy, modify, and distribute this
 * software and its documentation for NON-COMMERCIAL purposes
 * and without fee is hereby granted provided that this
 * copyright notice appears in all copies.
 *
 * THE AUTHORS AND PUBLISHER MAKE NO REPRESENTATIONS OR
 * WARRANTIES ABOUT THE SUITABILITY OF THE SOFTWARE, EITHER
 * EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A
 * PARTICULAR PURPOSE, OR NON-INFRINGEMENT. THE AUTHORS
 * AND PUBLISHER SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED
 * BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING
 * THIS SOFTWARE OR ITS DERIVATIVES.
 */

import java.io.*;


/* Format the number following printf conventions. */

public class IOFormat {
 private int width;
 private int precision;
 private String pre;
 private String post;
 private boolean leading_zeroes;
 private boolean show_plus;
 private boolean alternate;
 private boolean show_space;
 private boolean left_align;
 private char fmt;


 /*  The constructor initializes IOFormat. */

 public IOFormat(String s) {
  width = 0;
  precision = -1;
  pre = "";
  post = "";
  leading_zeroes = false;
  show_plus = false;
  alternate = false;
  show_space = false;
  left_align = false;
  fmt = ' ';

  int state = 0;
  int length = s.length();
  int parse_state = 0;
  // 0 = prefix, 1 = flags, 2 = width, 3 = precision,
  // 4 = format, 5 = end
  int i = 0;

  while (parse_state == 0) {
   if (i >= length)
    parse_state = 5;
```

```
  else if (s.charAt(i) == '%') {
   if (i < length - 1) {
    if (s.charAt(i + 1) == '%') {
     pre = pre + '%';
     i++;
    }
    else
     parse_state = 1;
   }
   else
    throw new java.lang.IllegalArgumentException();
  }
  else
   pre = pre + s.charAt(i);
  i++;
 }

 while (parse_state == 1) {
  if (i >= length)
   parse_state = 5;
  else if (s.charAt(i) == ' ')
   show_space = true;
  else if (s.charAt(i) == '-')
   left_align = true;
  else if (s.charAt(i) == '+')
   show_plus = true;
  else if (s.charAt(i) == '0')
   leading_zeroes = true;
  else if (s.charAt(i) == '#')
   alternate = true;
  else {
   parse_state = 2;
   i--;
  }
  i++;
 }

 while (parse_state == 2) {
  if (i >= length)
   parse_state = 5;
  else if ('0' <= s.charAt(i) && s.charAt(i) <= '9') {
   width = width * 10 + s.charAt(i) - '0';
   i++;
  }
  else if (s.charAt(i) == '.') {
   parse_state = 3;
   precision = 0;
   i++;
  }
  else
   parse_state = 4;
 }

 while (parse_state == 3) {
  if (i >= length)
   parse_state = 5;
  else if ('0' <= s.charAt(i) && s.charAt(i) <= '9') {
   precision = precision * 10 + s.charAt(i) - '0';
   i++;
  }
  else
   parse_state = 4;
 }

 if (parse_state == 4) {
  if (i >= length)
   parse_state = 5;
  else
   fmt = s.charAt(i);
  i++;
```

```
    }

    if (i < length)
      post = s.substring(i, length);
  }


  /* Print a formatted number following printf conventions
   * @param s a PrintStream
   * @param fmt the format string
   * @param x the double to print */

  public static void print(java.io.PrintStream s, String fmt, double x) {
    s.print(new IOFormat(fmt).form(x));
  }


  /* Print a formatted number following printf conventions
   * @param s a PrintStream
   * @param fmt the format string
   * @param x the long to print */

  public static void print(java.io.PrintStream s, String fmt, long x) {
    s.print(new IOFormat(fmt).form(x));
  }


  /* Print a formatted number following printf conventions
   * @param s a PrintStream, fmt the format string
   * @param x a string that represents the digits to print */

  public static void print(java.io.PrintStream s, String fmt, String x) {
    s.print(new IOFormat(fmt).form(x));
  }


  /* Convert a string of digits (decimal, octal or hex) to an integer
   * @param s a string
   * @return the numeric value of the prefix of s representing
   * a base 10 integer */

  public static int atoi(String s) {
    return (int)atol(s);
  }


  /* Convert a string of digits (decimal, octal or hex) to a long integer
   * @param s a string
   * @return the numeric value of the prefix of s representing
   * a base 10 integer */

  public static long atol(String s) {
    int i = 0;

    while (i < s.length() && Character.isSpace(s.charAt(i)))
      i++;

    if (i < s.length() && s.charAt(i) == '0') {
      if (i + 1 < s.length() && (s.charAt(i + 1) == 'x' || s.charAt(i + 1) == 'X'))
        return parseLong(s.substring(i + 2), 16);
      else
        return parseLong(s, 8);
    }
    else return parseLong(s, 10);
  }


  /* Parse a string of digits (decimal, octal or hex) to a long integer
   * @param s a string
   * @param base the decimal, octal or hex base
```

```
   * @return the numeric value of the prefix of s representing
   * a base 10 integer */

  private static long parseLong(String s, int base) {
   int i = 0;
   int sign = 1;
   long r = 0;

   while (i < s.length() && Character.isSpace(s.charAt(i)))
    i++;

   if (i < s.length() && s.charAt(i) == '-') {
    sign = -1;
    i++;
   }
   else if (i < s.length() && s.charAt(i) == '+') {
    i++;
   }

   while (i < s.length()) {
    char ch = s.charAt(i);
    if ('0' <= ch && ch < '0' + base)
     r = r * base + ch - '0';
    else if ('A' <= ch && ch < 'A' + base - 10)
     r = r * base + ch - 'A' + 10 ;
    else if ('a' <= ch && ch < 'a' + base - 10)
     r = r * base + ch - 'a' + 10 ;
    else
     return r * sign;
    i++;
   }
   return r * sign;
  }


  /* Format a double into a string (like sprintf in C)
   * @param x the number to format
   * @return the formatted string
   * @exception IllegalArgumentException if bad argument */

  public String form(double x) {
   String r;
   if (precision < 0)
    precision = 6;
   int s = 1;
   if (x < 0) {
    x = -x;
    s = -1;
   }
   if (fmt == 'f')
     r = fixed_format(x);
   else if (fmt == 'e' || fmt == 'E' || fmt == 'g' || fmt == 'G')
     r = exp_format(x);
   else
     throw new java.lang.IllegalArgumentException();

   return pad(sign(s, r));
  }


  /* Format a long integer into a string (like sprintf in C)
   * @param x the number to format
   * @return the formatted string */

  public String form(long x) {
   String r;
   int s = 0;
   if (fmt == 'd' || fmt == 'i') {
    s = 1;
    if (x < 0) {
```

```
      x = -x;
      s = -1;
     }
     r = "" + x;
    }
    else if (fmt == 'o')
     r = convert(x, 3, 7, "01234567");
    else if (fmt == 'x')
     r = convert(x, 4, 15, "0123456789abcdef");
    else if (fmt == 'X')
     r = convert(x, 4, 15, "0123456789ABCDEF");
    else
     throw new java.lang.IllegalArgumentException();

    return pad(sign(s, r));
  }


  /* Format a string into a larger string (like sprintf in C)
   * @param x the value to format
   * @return the formatted string */

  public String form(String s) {
    if (fmt != 's')
     throw new java.lang.IllegalArgumentException();
    if (precision >= 0)
     s = s.substring(0, precision);
    return pad(s);
  }


  //
  private static String repeat(char c, int n) {
    if (n <= 0)
     return "";
    StringBuffer s = new StringBuffer(n);
    for (int i = 0; i < n; i++)
     s.append(c);
    return s.toString();
  }


  //
  private static String convert(long x, int n, int m, String d) {
    if (x == 0)
     return "0";
    String r = "";
    while (x != 0) {
     r = d.charAt((int)(x & m)) + r;
     x = x >>> n;
    }
    return r;
  }


  //
  private String pad(String r) {
    String p = repeat(' ', width - r.length());
    if (left_align)
     return pre + r + p + post;
    else
     return pre + p + r + post;
  }


  //
  private String sign(int s, String r) {
    String p = "";
    if (s < 0)
     p = "-";
```

```
  else if (s > 0) {
   if (show_plus)
    p = "+";
   else if (show_space)
    p = " ";
  }
  else {
   if (fmt == 'o' && alternate && r.length() > 0 && r.charAt(0) != '0')
    p = "0";
   else if (fmt == 'x' && alternate)
    p = "0x";
   else if (fmt == 'X' && alternate)
    p = "0X";
  }
  int w = 0;
  if (leading_zeroes)
   w = width;
  else if ((fmt == 'd' || fmt == 'i' || fmt == 'x' || fmt == 'X' || fmt == 'o')
    && precision > 0)
   w = precision;

  return p + repeat('0', w - p.length() - r.length()) + r;
 }


 //
 private String fixed_format(double d) {
  String f = "";

  if (d > 0x7FFFFFFFFFFFFFFFL)
   return exp_format(d);

  long l = (long)(precision == 0 ? d + 0.5 : d);
  f = f + l;

  double fr = d - l; // fractional part
  if (fr >= 1 || fr < 0)
   return exp_format(d);

  return f + frac_part(fr);
 }


 //
 private String frac_part(double fr) {
  // precondition: 0 <= fr < 1
  String z = "";
  if (precision > 0) {
   double factor = 1;
   String leading_zeroes = "";
   for (int i = 1; i <= precision && factor <= 0x7FFFFFFFFFFFFFFFL; i++) {
    factor *= 10;
    leading_zeroes = leading_zeroes + "0";
   }
   long l = (long) (factor * fr + 0.5);

   z = leading_zeroes + l;
   z = z.substring(z.length() - precision, z.length());
  }

  if (precision > 0 || alternate)
   z = "." + z;
  if ((fmt == 'G' || fmt == 'g') && !alternate) {
  // remove trailing zeroes and decimal point
   int t = z.length() - 1;
   while (t >= 0 && z.charAt(t) == '0')
    t--;
   if (t >= 0 && z.charAt(t) == '.')
    t--;
   z = z.substring(0, t + 1);
```

```
    }
    return z;
  }


  //
  private String exp_format(double d) {
    String f = "";
    int e = 0;
    double dd = d;
    double factor = 1;
    while (dd > 10) {
      e++;
      factor /= 10;
      dd = dd / 10;
    }
    while (dd < 1) {
      e--;
      factor *= 10;
      dd = dd * 10;
    }
    if ((fmt == 'g' || fmt == 'G') && e >= -4 && e < precision)
      return fixed_format(d);

    d = d * factor;
    f = f + fixed_format(d);

    if (fmt == 'e' || fmt == 'g')
      f = f + "e";
    else
      f = f + "E";

    String p = "000";
    if (e >= 0) {
      f = f + "+";
      p = p + e;
    }
    else {
      f = f + "-";
      p = p + (-e);
    }

    return f + p.substring(p.length() - 3, p.length());
  }


}
```

A.17. Interface.java

```
import java.io.*;
import java.util.*;
import java.awt.*;


/* This class is responsible for updating user interface according to
 * system execution status. */

public class Interface extends Thread{

  private OSS oss;                  // OS simulation.
  private ProcessMgr processmgr;    // Process manager.
  private PageMgr pagemgr;          // Page manager.
  private Bookkeeper bookkeeper;    // Bookkeeper.
  private UserProcessRequestQueue reqQ; // New process request queue.
  private Vector readyQ;            // Ready queue.
  private Vector sleepQ;            // Sleep queue.
  private Vector msgQ;              // Message waiting queue.
  private Vector semQ;              // Semaphore waiting queue.

  private boolean frozen;           // Frozen flag.

  private int oldCPUUtilization;    // Old CPU-utilization value.
  private int oldThroughput;        // Old throughput value.
  private int oldTurnaroundT;       // Old turn-around-time value.
  private int oldResponseT;         // Old response-time value.
  private int oldWaitingT;          // Old waiting-time value.
  private int oldPageFaultR;        // Old page-fault-rate value.
  private int oldFreeMemory;        // Old memory availability value.

  private int lblReadyQProcNumV;    // Ready queue length value.
  private int lblSleepQProcNumV;    // Sleep queue length value.
  private int lblMessageQProcNumV;  // Message queue length value.
  private int lblSemaphoreProcNumV; // Semaphore queue length value.

  // Lists.
  private java.awt.List ReadyList;     // Ready-queue list.
  private java.awt.List SleepList;     // Sleep-queue list.
  private java.awt.List SemaphoreList; // Semaphore-waiting-queue list.
  private java.awt.List MessageWaitList; // Message-waiting-queue list.
  private java.awt.List HistoryList;   // Execution-history list.

  // Labels.
  private java.awt.Label RunProcLabel; // Running-process label.
  private java.awt.Label lblCPUV;      // CPU-utilization label.
  private java.awt.Label lblThroughputV; // Throughput label.
  private java.awt.Label lblResponesTV; // Response-time label.
  private java.awt.Label lblTurnaroundTV;// Turn-around-time label.
  private java.awt.Label lblWaitTV;    // Waiting-time label.
  private java.awt.Label lblPageFaultV; // Page-fault-rate label.
  private java.awt.Label lblFreeMemoryV; // Page-fault-rate label.

  private java.awt.Label lblReadyQProcNum; // Ready queue length label.
  private java.awt.Label lblSleepQProcNum; // Sleep queue length label.
  private java.awt.Label lblMessageQProcNum; // Message queue length label.
  private java.awt.Label lblSemaphoreProcNum;// Semaphore queue length label.

  // Button.
  private java.awt.Button btnAdd;      // Add-new-process button.

  // Canvas.
  private PerformanceMonitor cvsCPU;      // CPU-utilization canvas.
  private PerformanceMonitor cvsThroughput; // Throughput canvas.
  private PerformanceMonitor cvsResponesT; // Response-time canvas.
  private PerformanceMonitor cvsTurnaroundT;// Turn-around-time canvas.
  private PerformanceMonitor cvsWaitT;    // Waiting-time canvas.
```

```
  private PerformanceMonitor cvsPageFault;  // Page-fault-rate canvas.
  private PerformanceMonitor cvsFreeMemory; // Memory-availability canvas.


  /* Initialize user interface thread. */

  public void initialize(
     OSS oss,
     ProcessMgr processmgr,
     PageMgr pagemgr,
     Bookkeeper bookkeeper,
     UserProcessRequestQueue reqQ,
     java.awt.Button btnAdd ) {

   this.oss = oss;
   this.processmgr = processmgr;
   this.pagemgr = pagemgr;
   this.bookkeeper = bookkeeper;
   this.reqQ = reqQ;

   this.btnAdd = btnAdd;

   frozen = false;

   readyQ = new Vector();
   sleepQ = new Vector();
   msgQ = new Vector();
   semQ = new Vector();

   // -1 means that actual values are not available yet.
   oldCPUUtilization = -1;
   oldThroughput = -1;
   oldTurnaroundT = -1;
   oldResponseT = -1;
   oldWaitingT = -1;
   oldPageFaultR = -1;
   oldFreeMemory = -1;

   lblReadyQProcNumV = 0;
   lblSleepQProcNumV = 0;
   lblMessageQProcNumV = 0;
   lblSemaphoreProcNumV = 0;
  }


 /* Set Lists. */

 public void setLists(
     java.awt.List ReadyList,
     java.awt.List SleepList,
     java.awt.List SemaphoreList,
     java.awt.List MessageWaitList,
     java.awt.List HistoryList ){

  this.ReadyList = ReadyList;
  this.SleepList = SleepList;
  this.SemaphoreList = SemaphoreList;
  this.MessageWaitList = MessageWaitList;
  this.HistoryList = HistoryList;

  ReadyList.clear();
  SleepList.clear();
  SemaphoreList.clear();
  MessageWaitList.clear();
  HistoryList.clear();
 }


 /* Set canvas. */
```

```
 public void setCanvas(
    PerformanceMonitor cvsCPU,
    PerformanceMonitor cvsThroughput,
    PerformanceMonitor cvsResponesT,
    PerformanceMonitor cvsTurnaroundT,
    PerformanceMonitor cvsWaitT,
    PerformanceMonitor cvsPageFault,
    PerformanceMonitor cvsFreeMemory ){

  this.cvsCPU = cvsCPU;
  this.cvsThroughput = cvsThroughput;
  this.cvsResponesT = cvsResponesT;
  this.cvsTurnaroundT = cvsTurnaroundT;
  this.cvsWaitT = cvsWaitT;
  this.cvsPageFault = cvsPageFault;
  this.cvsFreeMemory = cvsFreeMemory;

  // Set maximum number of samples.
  cvsCPU.setMaxVals(300);
  cvsThroughput.setMaxVals(300);
  cvsResponesT.setMaxVals(300);
  cvsTurnaroundT.setMaxVals(300);
  cvsWaitT.setMaxVals(300);
  cvsPageFault.setMaxVals(300);
  cvsFreeMemory.setMaxVals(300);

  // Set maximum values.
  cvsCPU.setMax(100);
  cvsThroughput.setMax(50);
  cvsResponesT.setMax(50);
  cvsTurnaroundT.setMax(280);
  cvsWaitT.setMax(50);
  cvsPageFault.setMax(100);
  cvsFreeMemory.setMax(100);
 }


/* Set labels. */

public void setLabels(
    java.awt.Label lblCPUV,
    java.awt.Label lblThroughputV,
    java.awt.Label lblResponesTV,
    java.awt.Label lblTurnaroundTV,
    java.awt.Label lblWaitTV,
    java.awt.Label lblPageFaultV,
    java.awt.Label lblFreeMemoryV,
    java.awt.Label RunProcLabel,
    java.awt.Label lblReadyQProcNum,
    java.awt.Label lblSleepQProcNum,
    java.awt.Label lblMessageQProcNum,
    java.awt.Label lblSemaphoreProcNum) {

  this.lblCPUV = lblCPUV;
  this.lblThroughputV = lblThroughputV;
  this.lblResponesTV = lblResponesTV;
  this.lblTurnaroundTV = lblTurnaroundTV;
  this.lblWaitTV = lblWaitTV;
  this.lblPageFaultV = lblPageFaultV;
  this.lblFreeMemoryV = lblFreeMemoryV;

  this.lblReadyQProcNum = lblReadyQProcNum;
  this.lblSleepQProcNum = lblSleepQProcNum;
  this.lblMessageQProcNum = lblMessageQProcNum;
  this.lblSemaphoreProcNum = lblSemaphoreProcNum;

  this.RunProcLabel = RunProcLabel;
  RunProcLabel.setText("Shutdown");
 }
```

```
/* This is the control part of user interface thread. */

public void run(){

  while (true) {
   if (!frozen) {
    updateInterface(); // Update user interface.
    try{
     sleep(10);      // Be nice to other threads.
    } catch (InterruptedException e) { };
   }
   else{
    while (frozen)
     try{
      sleep(1000);
     } catch (InterruptedException e) { };
   }
  }
}


/* Update user interface. */

public void updateInterface(){

 // For interactive mode, add-new-process button is disabled when the
 // number of processes in the system reaches a maximum value.
 if (ConfigCurrent.MODE == EnumSimulationMode.INTERACTIVE)
  if (processmgr.getProcessNum()+reqQ.size() == ConfigCurrent.MAXPROCESS)
   btnAdd.disable();
  else
   btnAdd.enable();

 // For non-interactive mode, when there is no process running in
 // the system, the simulation ends.
 if (ConfigCurrent.MODE == EnumSimulationMode.NON_INTERACTIVE)
  if (processmgr.getProcessNum()+reqQ.size() == 0 ||
     (reqQ.size() == 0 && processmgr.getProcessNum() == msgQ.size()))
   oss.endSimulation();

 // Update performance monitor (CPU utilization) canvas.
 if (!frozen) {
  int iCPUUtilization = bookkeeper.computeCPUUtilization();
  if (iCPUUtilization != oldCPUUtilization){
   lblCPUV.setText("    ");
   lblCPUV.setText(iCPUUtilization + " %");
   oldCPUUtilization = iCPUUtilization;
  }
  cvsCPU.addVal(iCPUUtilization);
  cvsCPU.repaint();
 }

 // Update performance monitor (throughput) canvas.
 if (!frozen) {
  int iThroughput = bookkeeper.computeThroughput( (new Date()).getTime());
  if (iThroughput != oldThroughput){
   lblThroughputV.setText("    ");
   lblThroughputV.setText(iThroughput + "");
   oldThroughput = iThroughput;
  }
  cvsThroughput.addVal(iThroughput);
  cvsThroughput.repaint();
 }

 // Update performance monitor (response time) canvas.
 if (!frozen) {
  int iResponseT = bookkeeper.computeResponseT();
  if (iResponseT != -1){
   if (iResponseT != oldResponseT){
```

```
      lblResponesTV.setText("    ");
      lblResponesTV.setText(iResponseT + "");
      oldResponseT = iResponseT;
     }
     cvsResponesT.addVal(iResponseT);
     cvsResponesT.repaint();
    }
    else{
     lblResponesTV.setText(" NA");
    }
   }

   // Update performance monitor (turn around time) canvas.
   if (!frozen) {
    int iTurnaroundT = bookkeeper.computeTurnaroundT();
    if (iTurnaroundT != -1){
     if (iTurnaroundT != oldTurnaroundT){
      lblTurnaroundTV.setText("    ");
      lblTurnaroundTV.setText(iTurnaroundT + "");
      oldTurnaroundT = iTurnaroundT;
     }
     cvsTurnaroundT.addVal(iTurnaroundT);
     cvsTurnaroundT.repaint();
    }
    else{
     lblTurnaroundTV.setText(" NA");
    }
   }

   // Update performance monitor (waiting time) canvas.
   if (!frozen) {
    int iWaitingT = bookkeeper.computeWaitingT();
    if (iWaitingT != -1){
     if (iWaitingT != oldWaitingT){
      lblWaitTV.setText("    ");
      lblWaitTV.setText(iWaitingT + "");
      oldWaitingT = iWaitingT;
     }
     cvsWaitT.addVal(iWaitingT);
     cvsWaitT.repaint();
    }
    else{
     lblWaitTV.setText(" NA");
    }
   }

   // Update performance monitor (page fault rate) canvas.
   if (!frozen) {
    int iPageFaultR = bookkeeper.computePageFaultR();
    if (iPageFaultR != oldPageFaultR){
     lblPageFaultV.setText("    ");
     lblPageFaultV.setText(iPageFaultR + " %");
     oldPageFaultR = iPageFaultR;
    }
    cvsPageFault.addVal(iPageFaultR);
    cvsPageFault.repaint();
   }

   // Update memory availability canvas.
   if (!frozen) {
    int iFreeMemory = pagemgr.getFreeFrameNumber()*100
                /ConfigCurrent.MEMORYFRAMES;
    if (iFreeMemory != oldFreeMemory){
     lblFreeMemoryV.setText("    ");
     lblFreeMemoryV.setText(iFreeMemory + " %");
     oldFreeMemory = iFreeMemory;
    }
    cvsFreeMemory.addVal(iFreeMemory);
    cvsFreeMemory.repaint();
   }
```

```
}

/* Freeze execution. */

public void freezeExec(){

 // Allow some time to freeze Machine completely first.
 try{
  sleep(100);
 } catch (InterruptedException e) { };

 frozen = true;
}

/* Un-freeze. */

public void unFreeze(){
 frozen = false;
}

/* Add process into ready queue. */

public void readyQAdd(PCB proc){
 readyQ.addElement("Process "+proc.getID());
 ReadyList.addItem("Process "+proc.getID());

 lblReadyQProcNumV++;
 lblReadyQProcNum.setText(" ");
 lblReadyQProcNum.setText(lblReadyQProcNumV+"");
}

/* Add process into sleep queue. */

public void sleepQAdd(PCB proc){
 sleepQ.addElement("Process "+proc.getID());
 SleepList.addItem("Process "+proc.getID());

 lblSleepQProcNumV++;
 lblSleepQProcNum.setText(" ");
 lblSleepQProcNum.setText(lblSleepQProcNumV+"");
}

/* Add process into message waiting queue. */

public void msgQAdd(PCB proc){
 msgQ.addElement("Process "+proc.getID());
 MessageWaitList.addItem("Process "+proc.getID());

 lblMessageQProcNumV++;
 lblMessageQProcNum.setText(" ");
 lblMessageQProcNum.setText(lblMessageQProcNumV+"");
}

/* Add process into semaphore waiting queue. */

public void semQAdd(PCB proc){
 semQ.addElement("Process "+proc.getID());
 SemaphoreList.addItem("Process "+proc.getID());

 lblSemaphoreProcNumV++;
 lblSemaphoreProcNum.setText(" ");
 lblSemaphoreProcNum.setText(lblSemaphoreProcNumV+"");
}
```

```
/* Remove process from ready queue. */

public void readyQRemove(PCB proc){
 ReadyList.delItem(readyQ.indexOf("Process "+proc.getID()));
 readyQ.removeElement("Process "+proc.getID());

 lblReadyQProcNumV--;
 lblReadyQProcNum.setText(" ");
 lblReadyQProcNum.setText(lblReadyQProcNumV+"");
}


/* Remove process from sleep queue. */

public void sleepQRemove(PCB proc){
 SleepList.delItem(sleepQ.indexOf("Process "+proc.getID()));
 sleepQ.removeElement("Process "+proc.getID());

 lblSleepQProcNumV--;
 lblSleepQProcNum.setText(" ");
 lblSleepQProcNum.setText(lblSleepQProcNumV+"");
}


/* Remove process from message waiting queue. */

public void msgQRemove(PCB proc){
 MessageWaitList.delItem(msgQ.indexOf("Process "+proc.getID()));
 msgQ.removeElement("Process "+proc.getID());

 lblMessageQProcNumV--;
 lblMessageQProcNum.setText(" ");
 lblMessageQProcNum.setText(lblMessageQProcNumV+"");
}


/* Remove process from semaphore waiting queue. */
public void semQRemove(PCB proc){
 SemaphoreList.delItem(semQ.indexOf("Process "+proc.getID()));
 semQ.removeElement("Process "+proc.getID());

 lblSemaphoreProcNumV--;
 lblSemaphoreProcNum.setText(" ");
 lblSemaphoreProcNum.setText(lblSemaphoreProcNumV+"");
}


/* Update the running process label. */

public void runningProcess(PCB proc){
 int id = proc.getID();
 RunProcLabel.setText("          ");
 if (id != ConfigCurrent.MAXPROCESS)
   RunProcLabel.setText("Process "+id);
 else
   RunProcLabel.setText("Null Process");
}


/* Print out system performance information. */

public void printout(PrintStream os) throws IOException{
 int iCPUUtilization = bookkeeper.computeCPUUtilization();
 int iThroughput = bookkeeper.computeThroughput( (new Date()).getTime());
 int iResponseT = bookkeeper.computeResponseT();
 int iTurnaroundT = bookkeeper.computeTurnaroundT();
 int iWaitingT = bookkeeper.computeWaitingT();
 int iPageFaultR = bookkeeper.computePageFaultR();
```

```
    IOFormat.print(os, "%s\n", "*************************************************");
    IOFormat.print(os, "%s\n", "*      SYSTEM PERFORMANCE INFORMATION       *");
    IOFormat.print(os, "%s\n", "*************************************************");

    IOFormat.print(os, "%s ---- ", "CPU utilization (%)");
    IOFormat.print(os, "%d\n", iCPUUtilization);
    IOFormat.print(os, "%s ---- ", "Throughput (number of processes per minute)");
    IOFormat.print(os, "%d\n", iThroughput);
    IOFormat.print(os, "%s ---- ", "Response Time (seconds)");
    IOFormat.print(os, "%d\n", iResponseT);
    IOFormat.print(os, "%s ---- ", "Turnaround Time (seconds)");
    IOFormat.print(os, "%d\n", iTurnaroundT);
    IOFormat.print(os, "%s ---- ", "Wait Time (seconds)");
    IOFormat.print(os, "%d\n", iWaitingT);
    IOFormat.print(os, "%s ---- ", "PageFault Rate (%)");
    IOFormat.print(os, "%d\n", iPageFaultR);

    IOFormat.print(os, "%s\n", " ");
  }


}
```

A.18. Machine.java

```
import java.io.*;
import java.util.*;


/* This class models the machine hardware which includes the registers,
 * the main memory, the backstore and the page table. */

class Machine {

 // General purpose registers. The registers may be used
 // as variables in user programs.
 private static byte[] registers = new byte[ConfigCurrent.NREGS+1];

 // Special purpose register.
 private static int PTBR;       // Page table base register.

 private static int faultingPage; // Used to store the faulting page number.
 private static Page[] pageTable; // Page table.

 // Main meory.
 private byte[] mainMemory = new byte[ConfigCurrent.MEMSIZE];

 // Backstore.
 private byte[] backStore = new byte[ConfigCurrent.BACKSTORESIZE];

 // DMA counter simulates the time needed to finish a DMA operation.
 private int DMACounter = 0;

 // Semaphore counter simulates the time needed to finish a manipulating
 // resource operation.
 private int[] semCounter = new int[ConfigCurrent.NUMSEM1 + ConfigCurrent.NUMSEM2];

 private Vector usedSemCounter = new Vector();// Semaphore counters vector.
 private DMARequest DMAReq;              // DMA request.
 private boolean DMAPending;             // DMA pending flag.
 private OS os;                          // OS object.
 private UserProcessRequestQueue reqQ;       // Request to create new process.
 private Bookkeeper bookkeeper;          // Bookkeeper.
 private boolean frozen;                 // Operation frozen flag.
 private Yield yield;                    // Yield execution to others.
 private boolean debugflag;              // Debugging flag.


/* Constructor initializes Machine. */

public Machine(
   UserProcessRequestQueue reqQ,
   Yield yield,
   Bookkeeper bookkeeper) {

 this.reqQ = reqQ;
 this.yield = yield;
 this.bookkeeper = bookkeeper;

 // Initialize registers.
 for ( int i = 1; i <= ConfigCurrent.NREGS; i++ )
  registers[i] = 0;

 // Initialize main memory.
 for ( int i = 0; i < ConfigCurrent.MEMSIZE; i++ )
  mainMemory[i] = 0;

 // Initialize backstore.
 for ( int i = 0; i < ConfigCurrent.BACKSTORESIZE; i++ )
  backStore[i] = 0;
```

```
  // Initialize page table.
  pageTable = new Page[ConfigCurrent.PAGEENTRY];
  for(int i = 0; i < ConfigCurrent.PAGEENTRY; i++){
   Page page = new Page(0, false, false, false);
   setPageEntry(i, page);
  }

  // Initialize semaphore counters.
  for(int i = 0; i < (ConfigCurrent.NUMSEM1 + ConfigCurrent.NUMSEM2); i++){
   semCounter[i] = 0;
  }

  frozen = false;

  DMAPending = false;
 }


/* Set OS. */

public void setOS(OS os) {
 this.os = os;
}


/* Copy data from main memory to a register. */

public void memory2Register (int vaddr, int reg) {

 debug("Load: page " + vaddr/ConfigCurrent.PAGESIZE + " reg " + reg + "\n");

 int page, offset, frame, paddr;

 if ( (vaddr >= ConfigCurrent.ADDRSPACESIZE) || (reg > ConfigCurrent.NREGS))
  generateInterrupt(EnumInterrupt.ILLEGALADDRESS);

 // The OS should terminate this process/thread,
 // that is we should not return here.

 // Do the address translation using page table.
 page = vaddr / ConfigCurrent.PAGESIZE;
 offset = vaddr % ConfigCurrent.PAGESIZE;

 // Check if the page is in memory or not.
 if (pageTable[page+PTBR].getValid() == false) {
  debug("Page Fault on (page, PTBR) (" + page + "," + PTBR + ")\n");

  faultingPage = page;          // Store the faulting page number.
  bookkeeper.addPageFaultR(true); // Store the sample.
  generateInterrupt(EnumInterrupt.PAGEFAULT); // Issue a page fault interrupt.
 }

 // At this point, the page should be in the main memory.
 bookkeeper.addPageFaultR(false);  // Store the sample.

 // Load the register and update the page table.
 frame = pageTable[page+PTBR].getMemFrame();
 paddr = frame * ConfigCurrent.PAGESIZE + offset;
 registers[reg] = mainMemory[paddr];
 pageTable[page+PTBR].setRef(true);
}


/* Generate software interrupt. */

public void generateSwInterrupt() {
 generateInterrupt(EnumInterrupt.SOFTWARE);
}
```

```
/* Simulated idle operation. */

public void idle() {
  while (true)
    try{
      tick(); // Advance time by one unit.
      yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException e) { };
}


/* Set semaphore counter. */

public void setSemCounter (int index, int value) {
  usedSemCounter.addElement(new Integer(index));
  semCounter[index] = value;
}


/* Disable semaphore counter. */

public void disableSemCounter (int index) {
  usedSemCounter.removeElement(new Integer(index));
}


/* Set up a DMA request. */

public void DMASetup (DMARequest req) {
  debug("DMASetup");
  DMAPending = true;
  DMAReq = req;                    // Store the request locally.
  DMACounter = ConfigCurrent.DMADELAY; // Delay before the IO completes.
}


/* Advance time by one unit. */

public void tick() {
  debug("Tick");

  // System execution speed control.
  try{
    yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };

  // Sleep when frozen.
  while (frozen)
    try{
      yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException e) { };

  // Used by forking new process operation.
  byte[] saveregs = new byte[ConfigCurrent.NREGS];
  int retvalue, index;

  // Update DMA counter.
  if (DMAPending) {
    debug("------ DMACOUNTER: "+DMACounter);
    DMACounter--;
  }

  // Update semaphore counter.
  for (Enumeration e = usedSemCounter.elements() ; e.hasMoreElements() ;) {
    Integer in = (Integer) e.nextElement();
    index = in.intValue();
    semCounter[index]--;
  }

  // First, generate clock interrupt.
```

```
    generateInterrupt (EnumInterrupt.CLOCK);

    // Now generate DMA interrupt, if needed
    debug(" ---------- CHECK DMACOUNTER ----------");
    if ( (DMAPending) && (DMACounter <= 0) ) {
     if ( DMAReq.getDirection() == EnumDMATransferDirection.MEMTOBS )
      mem2Bs (DMAReq.getMemFrame(), DMAReq.getBsFrame());
     else
      bs2Mem (DMAReq.getBsFrame(), DMAReq.getMemFrame());
     DMAPending = false;
     generateInterrupt(EnumInterrupt.DMA);
    }

    // Now generate resource done interrupt, if needed
    debug(" ---------- CHECK SEM COUNTERS ----------");
    for (Enumeration e = usedSemCounter.elements() ; e.hasMoreElements() ;) {
     Integer in = (Integer) e.nextElement();
     index = in.intValue();
     if (semCounter[index] <= 0){
      disableSemCounter(index);
      generateInterrupt(EnumInterrupt.RESOURCEDONE);
     }
    }

    // Check request queue and generate a software interrupt.
    while (!reqQ.empty()){

     // Fork related work is shown here.....
     System.arraycopy(registers, 0, saveregs, 0, ConfigCurrent.NREGS);

     registers[1] = (byte) EnumSystemCallNum.FORK;
     registers[2] = 0; // Program file name.
     registers[3] = 0; // Parameter 1.
     registers[4] = 0; // Parameter 2.

     generateSwInterrupt();
     retvalue = registers[8];

     System.arraycopy(saveregs, 0, registers, 0, ConfigCurrent.NREGS);
    }
   }


/* Set debugging flag. */

public void setDebug (boolean flag) {
 debugflag = flag;
}


/* Return registers. */

public byte[] getRegisters(){
 return registers;
}


/* Set registers. */

public void setRegisters(byte[] reg){
 System.arraycopy(reg, 0, registers, 0, reg.length);
}


/* Return PTBR. */

public int getPTBR(){
 return PTBR;
}
```

```
/* Set PTBR. */

public void setPTBR(int ptbr){
  PTBR = ptbr;
}


/* Return faulting page number. */

public int getFaultingPage(){
  return faultingPage;
}


/* Return DMA pending flag. */

public boolean isDMAPending(){
  return DMAPending;
}


/* Set page table entry. */

public void setPageEntry(int index, Page page){
  pageTable[index] = page;
}


/* Return page table entry. */

public Page getPageEntry(int index){
  return pageTable[index];
}


/* Freeze execution. */

public void freezeExec(){
  frozen = true;
}


/* Un-freeze execution. */

public void unFreeze(){
  frozen = false;
}


/* Generate interrupt. */

private void generateInterrupt (int num) {
  //Debug("GenInterrupt");
  os.interruptHandler(num);
}


/* Copy a page from backstore to main memory frame. */

private void bs2Mem (int bsframe, int memframe) {
  debug("BStoMem");

  int memaddr = memframe * ConfigCurrent.PAGESIZE;
  int bsaddr = bsframe * ConfigCurrent.PAGESIZE;

  for ( int i = 0; i < ConfigCurrent.PAGESIZE; i++ ) {
    mainMemory[memaddr] = backStore[bsaddr];
    memaddr++;
    bsaddr++;
```

```
  }
 }

 /* Copy a page from main memory frame to backstore. */

 private void mem2Bs (int memframe, int bsframe) {
  debug("MemtoBS");

  int memaddr = memframe * ConfigCurrent.PAGESIZE;
  int bsaddr = bsframe * ConfigCurrent.PAGESIZE;

  for ( int i = 0; i < ConfigCurrent.PAGESIZE; i++ ) {
   backStore[bsaddr] = mainMemory[memaddr];
   memaddr++;
   bsaddr++;
  }
 }


 /* Debugging. */

 private void debug (String str) {
  if (debugflag)
   System.out.println( "Machine: " + str );
 }


}
```

A.19. MemoryFrame.java

```
/* This class models a memory frame. */

public class MemoryFrame{

  private PCB process;  // PCB.
  private int memFrame; // Memory frame number.
  private int vpage;    // Virtual page number.
  private boolean free; // Flag.


  /* Constructor initializes MemoryFrame. */

  public MemoryFrame(boolean free, int memFrame, int vpage, PCB process){
    this.free = free;
    this.memFrame = memFrame;
    this.vpage = vpage;
    this.process = process;
  }


  /* Set PCB. */

  public void setPCB(PCB process){
    this.process = process;
  }


  /* Return PCB. */

  public PCB getPCB(){
    return process;
  }


  /* Set virtual page. */

  public void setVpage(int vpage){
    this.vpage = vpage;
  }


  /* Return virtual page. */

  public int getVpage(){
    return vpage;
  }


  /* Return memory frame. */

  public int getMemFrame(){
    return memFrame;
  }


}
```

A.20. Message.java

```
/* This class models a message passed among processes. */

public class Message{

  private PCB sender;     // Sender.
  private PCB receiver;   // Receiver.
  private String contents; // Message contents.


  /* Constructor initializes Message. */

  public Message(PCB sender, PCB receiver, String contents){
    this.sender = sender;
    this.receiver = receiver;
    this.contents = contents;
  }


  /* Return sender. */

  public PCB getSender(){
    return sender;
  }


}
```

A.21. MessageMgr.java

```
import java.io.*;
import java.util.*;


/* Message Management is responsible for the communications among processes.
 * Processes can communicate with each other only while they have the
 * same keys. */

public class MessageMgr{

 /* We use an array of vector to implement data structures for messages.
  * Each vector is actually a message queue. Every process must specify
  * a numeric key as the array index to designates which message queue to
  * use for sending or receiving a message. */

 private Vector[] messageQ;
 private Vector waitingMessageQ; // Store processes which are waiting for
                                 // a message.
 private ProcessMgr processmgr; // Porcess Management.
 private Scheduler scheduler;   // Scheduler.
 private Machine machine;       // Machine.
 private Yield yield;           // Yeild.
 private History history;       // History.
 private Interface userinterface; // Interface.
 private boolean updated;       // Updating flag of message waiting queue.
 private boolean debugflag;     // Debug flag.


 /* Constructor initializes Message Management. */

 MessageMgr(ProcessMgr p, Scheduler s, Machine m, Yield yield, History history,
  Interface userinterface){
  messageQ = new Vector[ConfigCurrent.NUMKEYS];
  for(int i = 0; i < ConfigCurrent.NUMKEYS; i++){
   messageQ[i] = new Vector();
  }
  waitingMessageQ = new Vector();
  processmgr = p;
  scheduler = s;
  machine = m;
  updated = false;
  this.yield = yield;
  this.history = history;
  this.userinterface = userinterface;
 }


 /* Message Management puts the sent message into the appropriate message
  * queue designated by the sending process through the key. Then checks
  * the message waiting queue to find out whether there is a process waiting
  * for a message from the queue. If so, removes the process from the waiting
  * queue and calls Scheduler to put it into the ready queue. */

 public int msgSnd(){
  if (debugflag)
   printMsgT();
  debug("Msgsnd");

  byte[] registers = machine.getRegisters();
  int key = registers[2];
  PCB sender = processmgr.getProcTable().currPro(registers[3]);
  PCB receiver = processmgr.getProcTable().currPro(registers[4]);
  Message msg = new Message(sender, receiver, null);
  messageQ[key].addElement(msg);
  PCB proc = null;
  if(debugflag)
```

```
    printMsgT();
   // clear sendmsgOp flag in UserProcess
   PCB curproc = processmgr.getCurrPro();
   UserProcess userpro = curproc.getUserPro();
   userpro.clearSendMsgOp();

   debug("messageQ.size: " + waitingMessageQ.size());
   if (debugflag)
    printMessageQ();
   for (Enumeration e = waitingMessageQ.elements() ; e.hasMoreElements() ;) {
    proc = (PCB) e.nextElement();
    if(proc.getKey() == key)
     break;
    else
     proc = null;
   }
   if (proc == null)
    return 0;
   else {
    waitingMessageQ.removeElement(proc);
    userinterface.msgQRemove(proc);
    history.writeHistory(proc, "gets the message it wanted");
    updated = true;
    try{
     yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException e) { };

    scheduler.makeReady(proc);
   }
   debug("after check, messageQ.size: " + waitingMessageQ.size());
   if (debugflag)
    printMessageQ();
   return 0;
}


/* Message Management goes through the appropriate message queue designated by
 * the receiving process to see whether there is a message not sent by itself.
 * If so, removes the message from that message queue. Otherwise, sets the
 * process to wait for a message from that queue, set its state to
 * MessageWait, and puts it into message waiting queue. */

public int msgRcv(){
 if (debugflag)
  printMsgT();
 debug("Msgrcv");
 byte[] registers = machine.getRegisters();
 PCB proc = processmgr.getCurrPro();
 int key = registers[2];
 Message msg;
 int id;

 for (Enumeration e = messageQ[key].elements() ; e.hasMoreElements() ;) {
  msg = (Message) e.nextElement();
  id = msg.getSender().getID();
  if (id != proc.getID()){
   messageQ[key].removeElement(msg);
   // clear receivemsgOp flag in UserProcess
   UserProcess userpro = proc.getUserPro();
   userpro.clearReceiveMsgOp();
   debug("messageQ.size: " + waitingMessageQ.size());
   if (debugflag) {
    printMessageQ();
    printMsgT();
   }
   return 0;
  }
 }

// must wait
```

```
    proc.setKey(key);
    proc.setState(EnumProcessState.MessageWait);
    waitingMessageQ.addElement(proc);
    userinterface.msgQAdd(proc);
    history.writeHistory(proc, "waits for the message it wants");
    updated = true;
    try{
     yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException e) { };

    debug("messageQ.size: " + waitingMessageQ.size());
    if (debugflag) {
     printMessageQ();
     printMsgT();
    }
    return 0;
  }


  /* Returns message waiting queue. */

  public Vector getMessageQ(){
   return waitingMessageQ;
  }


  /* Returns updating flag of message waiting queue. */

  public boolean messageQUpdated(){
   return updated;
  }


  /* Clears updating flag of message waiting queue. */

  public void clearUpdated(){
   updated = false;
  }


  /* Assigns debug flag. */

  public void setDebug (boolean flag) {
   debugflag = flag;
  }


  /* Prints out layout of message queues. */

  private void printMsgT(){
   Message msg;
   System.out.println("MessageTable:");
   for(int i=0; i<ConfigCurrent.NUMKEYS; i++){
    System.out.print("Key "+i+"->");
    for (Enumeration e = messageQ[i].elements() ; e.hasMoreElements() ;) {
     msg = (Message) e.nextElement();
     System.out.print("M,");
    }
    System.out.println("");
   }
  }


  /* Prints out message waiting queues. */

  private void printMessageQ(){
   PCB proc;
   System.out.print("Message WaitQ -> ");
   for (Enumeration e = waitingMessageQ.elements() ; e.hasMoreElements() ;) {
    proc = (PCB) e.nextElement();
```

```
    System.out.print("P"+proc.getID()+",");
   }
   System.out.println("");
  }

  /* Prints out a information line for debugging purpose. */
  private void debug (String str) {
   if (debugflag)
    System.out.println( "MessageMgr: " + str );
  }

}
```

A.22. OS.java

```
/* This class models the operating system level functionalities. */

public class OS {

 private Scheduler scheduler;      // Scheduler.
 private ProcessMgr processmgr;    // Process manager.
 private PageMgr pagemgr;          // Page manager.
 private Machine machine;          // Machine.
 private MessageMgr messagemgr;    // Message manager.
 private SemaphoreMgr semaphoremgr; // Semaphore manager.
 private ResourceMgr resourcemgr;  // Resource manager.
 private boolean debugflag;        // Debug flag.


 /* Constructor initializes OS. */

 public OS(
    Scheduler scheduler,
    ProcessMgr processmgr,
    PageMgr pagemgr,
    Machine machine,
    MessageMgr messagemgr,
    SemaphoreMgr semaphoremgr,
    ResourceMgr resourcemgr) {

  this.scheduler = scheduler;
  this.processmgr = processmgr;
  this.pagemgr = pagemgr;
  this.machine = machine;
  this.messagemgr =  messagemgr;
  this.semaphoremgr =  semaphoremgr;
  this.resourcemgr = resourcemgr;
 }


 /* Interrupt handler. */

 public void interruptHandler (int num) {

  switch(num){
   case EnumInterrupt.CLOCK:
    clockHandler();
    break;
   case EnumInterrupt.DMA:
    DMAHandler();
    break;
   case EnumInterrupt.PAGEFAULT:
    pageFaultHandler();
    break;
   case EnumInterrupt.ILLEGALADDRESS:
    illegalAddressHandler();
    break;
   case EnumInterrupt.SOFTWARE:
    softwareHandler();
    break;
   case EnumInterrupt.RESOURCEDONE:
    resourceReleaseHandler();
    break;
  }
 }


 /* Set debug flag. */

 public void setDebug (boolean flag) {
  debugflag = flag;
```

```
  }

  /* Clock interrupt handler. */

  private void clockHandler(){
   debug("ClockHandler");
   scheduler.run();
  }

  /* DMA interrupt handler. */

  private void DMAHandler(){
   debug("DMAHandler");
   processmgr.wakeup();
   scheduler.run();
  }

  /* Page-fault interrupt handler. */

  private void pageFaultHandler(){
   debug("PageFaultHandler");
   PCB pcb = processmgr.getCurrPro();
   int vpage = machine.getFaultingPage();
   int ptbr = machine.getPTBR();
   if (pagemgr.inFreeFrameList(pcb, vpage)){
    int memframe = (pagemgr.getMemFrame()).getMemFrame();
    Page page = machine.getPageEntry(ptbr + vpage);
    page.setMemFrame(memframe);
    page.setValid(true);
   }
   else{
    processmgr.sleep();
    scheduler.run();
   }
  }

  /* Illegal-address-access interrupt handler. */

  private void illegalAddressHandler(){
   debug("IllegalAddressHandler");
   processmgr.exit();
  }

  /* Software interrupt handler. */

  private void softwareHandler(){
   debug("SoftwareHandler");
   byte[] registers = machine.getRegisters();
   switch(registers[1]){
    case EnumSystemCallNum.FORK:
     registers[8] = (byte) processmgr.fork();
     break;
    case EnumSystemCallNum.GETPID:
     break;
    case EnumSystemCallNum.YIELD:
     break;
    case EnumSystemCallNum.MSGSND:
     registers[8] = (byte) messagemgr.msgSnd();
     break;
    case EnumSystemCallNum.MSGRCV:
     registers[8] = (byte) messagemgr.msgRcv();
     break;
    case EnumSystemCallNum.SemPop:
     registers[8] = (byte) semaphoremgr.semPop();
     break;
```

```
      case EnumSystemCallNum.SemVop:
        registers[8] = (byte) semaphoremgr.semVop();
        break;
    }
    scheduler.run();
  }


  /* Resource-done interrupt handler. */

  private void resourceReleaseHandler(){
    resourcemgr.releaseRes();
    scheduler.run();
  }


  /* Debug. */

  private void debug (String str) {
    if (debugflag)
      System.out.println( "OS: " + str );
  }


}
```

A.23. OSS.java

```
import java.awt.*;
import java.lang.*;


/* This class is the starting point of the system. It draws the user interface
 * and transfers user's interactions to proper classes for execution. */

public class OSS extends Frame implements ResultProcessor {

 private static Simulation simulation;     // Simulation.
 private static Configuration configuration; // Configuration.
 private static Help help;                 // Help;
 private static boolean bFrozen;           // Flag.

 private static Yield yield;               // Yield execution.

 // Buttons.
 private static java.awt.Button btnAdd;    // Add-new-process button.
 private static java.awt.Button btnEnd;    // End-simulation button.
 private static java.awt.Button btnStart;  // Start-simulation button.
 private static java.awt.Button btnConfig; // Configuration button.
 private static java.awt.Button btnFreeze; // Freeze-simulation button.
 private static java.awt.Button btnSlower; // Decrease-speed button.
 private static java.awt.Button btnFaster; // Increase-speed button.
 private static java.awt.Button btnHelp;   // Help button.
 private static java.awt.Button btnShutdown; // Shut-down-system button.

 // Lists.
 private static java.awt.List SleepList;   // Sleep-queue list.
 private static java.awt.List MsgWaitList; // Message-waiting-queue list.
 private static java.awt.List SemaphoreList;// Semaphore-waiting-queue list.
 private static java.awt.List ReadyList;   // Ready-queue list.
 private static java.awt.List HistoryList; // Execution-history list.

 // Canvas.
 private static PerformanceMonitor cvsCPU;      // CPU utilization canvas.
 private static PerformanceMonitor cvsThroughput; // Throughput canvas.
 private static PerformanceMonitor cvsResponesT; // Response-time canvas.
 private static PerformanceMonitor cvsTurnaroundT;// Turn-around-time canvas.
 private static PerformanceMonitor cvsWaitT;    // Waiting-time canvas.
 private static PerformanceMonitor cvsPageFault; // Page-fault-rate canvas.
 private static PerformanceMonitor cvsFreeMemory; // Free-memory canvas.

 // Labels.
 private static java.awt.Label blank;
 private static java.awt.Label lblTitle;     // Title.
 private static java.awt.Label lblRunning;
 private static java.awt.Label lblRunningProc; // Running process.
 private static java.awt.Label lblReadyQ;
 private static java.awt.Label lblSleepQ;
 private static java.awt.Label lblMessageQ;
 private static java.awt.Label lblSemaphore;
 private static java.awt.Label lblReadyQProcNum;  // Queue length.
 private static java.awt.Label lblSleepQProcNum;
 private static java.awt.Label lblMessageQProcNum;
 private static java.awt.Label lblSemaphoreProcNum;
 private static java.awt.Label lblCPU;
 private static java.awt.Label lblCPU1;
 private static java.awt.Label lblCPUV;      // Cpu utilization.
 private static java.awt.Label lblThroughput;
 private static java.awt.Label lblThroughputV; // Throughput.
 private static java.awt.Label lblResponesT;
 private static java.awt.Label lblResponesT1;
 private static java.awt.Label lblResponesTV; // Response time.
 private static java.awt.Label lblTurnaroundT;
 private static java.awt.Label lblTurnaroundT1;
```

```
private static java.awt.Label lblTurnaroundTV;// Turn around time.
private static java.awt.Label lblWaitT;
private static java.awt.Label lblWaitT1;
private static java.awt.Label lblWaitTV;     // Waiting time.
private static java.awt.Label lblPageFault;
private static java.awt.Label lblPageFault1;
private static java.awt.Label lblPageFaultV;  // Page fault rate.
private static java.awt.Label lblFreeMemory;
private static java.awt.Label lblFreeMemory1;
private static java.awt.Label lblFreeMemoryV; // Memory availability.
private static java.awt.Label lblHistory;


/* Constructor initializes OSS. */

public OSS(){

 // Initialize configuration window.
 configuration = new Configuration(this);
 configuration.resize(880,400);

 // Initialize help window.
 help = new Help();
 help.resize(600, 500);

 yield = new Yield();

 // Layout management constants.
 int none = GridBagConstraints.NONE;
 int hori = GridBagConstraints.HORIZONTAL;
 int vert = GridBagConstraints.VERTICAL;
 int both = GridBagConstraints.BOTH;
 int center = GridBagConstraints.CENTER;
 int north = GridBagConstraints.NORTH;
 int northeast = GridBagConstraints.NORTHEAST;
 int east = GridBagConstraints.EAST;

 setTitle("Operating System Simulation (OSS) In Java");

 GridBagLayout gbl = new GridBagLayout();
 setLayout(gbl);

 // Initialize buttons.
 btnAdd = new java.awt.Button("Add");
 btnAdd.disable();
 btnEnd = new java.awt.Button("End");
 btnEnd.disable();
 btnStart = new java.awt.Button("Start");
 btnConfig = new java.awt.Button("Config");
 btnFreeze = new java.awt.Button("Freeze");
 btnFreeze.disable();
 btnSlower = new java.awt.Button("Slower");
 btnSlower.disable();
 btnFaster = new java.awt.Button("Faster");
 btnFaster.disable();
 btnHelp = new java.awt.Button("Help");
 btnShutdown = new java.awt.Button("Shutdown");

 // Initialize labels.
 blank = new Label("      ");
 lblTitle = new Label("Operating System Simulation (OSS) In Java");
 lblTitle.setFont(new Font("Title", Font.BOLD, 18));
 lblTitle.setForeground(new Color(0));
 lblTitle.setBackground(new Color(12632256));

 lblRunning = new java.awt.Label("Current Running Process:");
 lblRunning.setFont(new Font("Running", Font.PLAIN, 14));

 lblRunningProc = new java.awt.Label("Shutdown");
 lblRunningProc.setFont(new Font("Dialog", Font.PLAIN, 16));
```

```
lblRunningProc.setForeground(new Color(0));
lblRunningProc.setBackground(new Color(12632256));

lblReadyQ = new Label("Ready Queue");
lblSleepQ = new Label("Sleep Queue");
lblMessageQ = new Label("Message Queue");
lblSemaphore = new Label("Semaphore Queue");

lblReadyQProcNum = new Label("0");
lblSleepQProcNum = new Label("0");
lblMessageQProcNum = new Label("0");
lblSemaphoreProcNum = new Label("0");

lblCPU = new Label("CPU");
lblCPU1 = new Label("Utilization");
lblCPUV = new Label("0 %");
lblThroughput = new Label("Throughput");
lblThroughputV = new Label("0");
lblResponesT = new Label("Response");
lblResponesT1 = new Label("Time");
lblResponesTV = new Label("NA");
lblTurnaroundT = new Label("Turnaround");
lblTurnaroundT1 = new Label("Time");
lblTurnaroundTV = new Label("NA");
lblWaitT = new Label("Waiting");
lblWaitT1 = new Label("Time");
lblWaitTV = new Label("NA");
lblPageFault = new Label("Page Fault");
lblPageFault1 = new Label("Rate");
lblPageFaultV = new Label("0 %");
lblFreeMemory = new Label("Memory");
lblFreeMemory1 = new Label("Availability");
lblFreeMemoryV = new Label("100 %");

lblHistory = new Label("Execution History");

// Initialize lists.
SleepList = new java.awt.List(0,false);
MsgWaitList = new java.awt.List(0,false);
SemaphoreList = new java.awt.List(0,false);
ReadyList = new java.awt.List(0,false);
HistoryList = new java.awt.List(0,false);

// Initialize canvas.
cvsCPU = new PerformanceMonitor();
cvsCPU.reshape(18,204,100,66);

cvsThroughput = new PerformanceMonitor();
cvsThroughput.reshape(18,204,100,66);
cvsResponesT = new PerformanceMonitor();
cvsResponesT.reshape(18,204,100,66);
cvsTurnaroundT = new PerformanceMonitor();
cvsTurnaroundT.reshape(18,204,100,66);
cvsWaitT = new PerformanceMonitor();
cvsWaitT.reshape(18,204,100,66);
cvsPageFault = new PerformanceMonitor();
cvsPageFault.reshape(18,204,100,66);
cvsFreeMemory = new PerformanceMonitor();
cvsFreeMemory.reshape(18,204,100,66);

cvsCPU.setBackground(new Color(0));
cvsThroughput.setBackground(new Color(0));
cvsResponesT.setBackground(new Color(0));
cvsTurnaroundT.setBackground(new Color(0));
cvsWaitT.setBackground(new Color(0));
cvsPageFault.setBackground(new Color(0));
cvsFreeMemory.setBackground(new Color(0));

// Lay down the user interface.
GridBagConstraints gbc = new GridBagConstraints();
```

```
    add(lblRunning, gbl, gbc, 0, 1, 3, 1, none, center, 0, 0);
    add(lblRunningProc, gbl, gbc, 3, 1, 6, 1, both, center, 20, 20);

    add(lblReadyQ, gbl, gbc, 0, 2, 2, 1, none, center, 0, 0);
    add(lblReadyQProcNum, gbl, gbc, 2, 2, 1, 1, none, center, 0, 0);
    add(lblSleepQ, gbl, gbc, 3, 2, 2, 1, none, center, 0, 0);
    add(lblSleepQProcNum, gbl, gbc, 5, 2, 1, 1, none, center, 0, 0);
    add(lblMessageQ, gbl, gbc, 8, 2, 2, 1, none, center, 0, 0);
    add(lblMessageQProcNum, gbl, gbc, 10, 2, 1, 1, none, center, 0, 0);
    add(lblSemaphore, gbl, gbc, 11, 2, 2, 1, none, center, 0, 0);
    add(lblSemaphoreProcNum, gbl, gbc, 13, 2, 1, 1, none, center, 0, 0);

    add(ReadyList, gbl, gbc, 0, 3, 3, 3, none, center, 100, 10);
    add(SleepList, gbl, gbc, 3, 3, 3, 3, none, center, 100, 10);
    add(MsgWaitList, gbl, gbc, 8, 3, 3, 3, none, center, 100, 10);
    add(SemaphoreList, gbl, gbc, 11, 3, 3, 3, none, center, 100, 10);

    add(blank, gbl, gbc, 0, 8, 14, 1, vert, center, 10, 10);

    add(lblCPU, gbl, gbc, 0, 9, 1, 1, none, center, 0, 0);
    add(lblCPU1, gbl, gbc, 0, 10, 1, 1, none, center, 0, 0);
    add(lblCPUV, gbl, gbc, 1, 9, 1, 2, none, center, 0, 0);
    add(lblThroughput, gbl, gbc, 2, 9, 1, 2, none, center, 0, 0);
    add(lblThroughputV, gbl, gbc, 3, 9, 1, 2, none, center, 0, 0);
    add(lblResponesT, gbl, gbc, 4, 9, 1, 1, none, center, 0, 0);
    add(lblResponesT1, gbl, gbc, 4, 10, 1, 1, none, center, 0, 0);
    add(lblResponesTV, gbl, gbc, 5, 9, 1, 2, none, center, 0, 0);
    add(lblTurnaroundT, gbl, gbc, 6, 9, 1, 1, none, center, 0, 0);
    add(lblTurnaroundT1, gbl, gbc, 6, 10, 1, 1, none, center, 0, 0);
    add(lblTurnaroundTV, gbl, gbc, 7, 9, 1, 2, none, center, 0, 0);
    add(lblWaitT, gbl, gbc, 8, 9, 1, 1, none, center, 0, 0);
    add(lblWaitT1, gbl, gbc, 8, 10, 1, 1, none, center, 0, 0);
    add(lblWaitTV, gbl, gbc, 9, 9, 1, 2, none, center, 0, 0);
    add(lblPageFault, gbl, gbc, 10, 9, 1, 1, none, center, 0, 0);
    add(lblPageFault1, gbl, gbc, 10, 10, 1, 1, none, center, 0, 0);
    add(lblPageFaultV, gbl, gbc, 11, 9, 1, 2, none, center, 0, 0);
    add(lblFreeMemory, gbl, gbc, 12, 9, 1, 1, none, center, 0, 0);
    add(lblFreeMemory1, gbl, gbc, 12, 10, 1, 1, none, center, 0, 0);
    add(lblFreeMemoryV, gbl, gbc, 13, 9, 1, 2, none, center, 0, 0);

    add(cvsCPU, gbl, gbc, 0, 11, 2, 8, none, center, 20, 20);
    add(cvsThroughput, gbl, gbc, 2, 11, 2, 8, none, center, 20, 20);
    add(cvsResponesT, gbl, gbc, 4, 11, 2, 8, none, center, 20, 20);
    add(cvsTurnaroundT, gbl, gbc, 6, 11, 2, 8, none, center, 20, 20);
    add(cvsWaitT, gbl, gbc, 8, 11, 2, 8, none, center, 20, 20);
    add(cvsPageFault, gbl, gbc, 10, 11, 2, 8, none, center, 20, 20);
    add(cvsFreeMemory, gbl, gbc, 12, 11, 2, 8, none, center, 20, 20);

    add(lblHistory, gbl, gbc, 0, 20, 14, 1, none, center, 0, 0);
    add(HistoryList, gbl, gbc, 0, 21, 14, 5, both, center, 0, 0);

    Panel buttons = new Panel();
    buttons.setLayout(new GridLayout(1,8));
    buttons.add(btnAdd);
    buttons.add(btnStart);
    buttons.add(btnConfig);
    buttons.add(btnFreeze);
    buttons.add(btnSlower);
    buttons.add(btnFaster);
    buttons.add(btnEnd);
    buttons.add(btnHelp);
    buttons.add(btnShutdown);
    add(buttons, gbl, gbc, 0, 0, 14, 1, hori, center, 0, 0);
  }


  /* Handle user's interactions with the interface. */

  public boolean handleEvent(Event event){
```

```
// A click on start-simulation button.
if(event.target == btnStart && event.id == Event.END){

 lblReadyQProcNum.setText("0");
 lblSleepQProcNum.setText("0");
 lblMessageQProcNum.setText("0");
 lblSemaphoreProcNum.setText("0");

 // Initialize simulation thread.
 simulation = new Simulation();

 // Pass the visual components that needs to be updated by
 // simulation thread.
 simulation.setVisualComponents(
  cvsCPU,cvsThroughput,cvsResponesT,cvsTurnaroundT,
  cvsWaitT,cvsPageFault,cvsFreeMemory,ReadyList,SleepList,SemaphoreList,
  MsgWaitList,HistoryList,lblCPUV,lblThroughputV,
  lblResponesTV,lblTurnaroundTV,lblWaitTV,lblPageFaultV,lblFreeMemoryV,
  lblRunningProc,lblReadyQProcNum,lblSleepQProcNum,
  lblMessageQProcNum,lblSemaphoreProcNum,btnAdd,this);

 // Start simulation.
 simulation.start();

 // Interactive simulation.
 if (ConfigCurrent.MODE == EnumSimulationMode.INTERACTIVE){
  setTitle("Operating System Simulation (OSS) In Java -- Interactive");
  lblRunningProc.setText("Null Process");
  btnEnd.enable();
  btnAdd.enable();
  btnStart.disable();
  btnFaster.enable();
  btnSlower.enable();
  btnFreeze.enable();
  btnConfig.disable();
 }

 // Non-interactive simulation.
 else{
  setTitle("Operating System Simulation (OSS) In Java -- Non-interactive");
  btnStart.disable();
  btnFaster.enable();
  btnSlower.enable();
  btnFreeze.enable();
  btnConfig.disable();
 }
 return(true);
}

// A click on configuration button.
else if(event.target == btnConfig && event.id == Event.END){

 // Pop up configuration window.
 configuration.show();
 return(true);
}

// A click on add-new-process button.
else if(event.target == btnAdd && event.id == Event.END){
 UserProcessRequest userReq = new UserProcessRequest(true,true,true,0);
 UserProcessRequestDialog pd = new UserProcessRequestDialog(this, userReq);
 pd.show();
 return true;
}

// A click on shut-down-system button.
else if(event.target == btnShutdown && event.id == Event.END){
 configuration.writeData();
 System.exit(0);
```

```
    return(true);
   }

   // A click on end-simulation button.
   else if(event.target == btnEnd && event.id == Event.END){
    endSimulation();
    return(true);
   }

   // A click on help button.
   else if(event.target == btnHelp && event.id == Event.END){
    help.show();
    return(true);
   }

   // A click on increase-simulation-speed button.
   else if(event.target == btnFaster && event.id == Event.END){
    ConfigCurrent.WAIT_TIME = ConfigCurrent.WAIT_TIME - ConfigCurrent.WAIT_TIME_PORTION;
    if (ConfigCurrent.WAIT_TIME < ConfigCurrent.WAIT_TIME_MIN){
     btnFaster.disable();
     ConfigCurrent.WAIT_TIME = ConfigCurrent.WAIT_TIME_MIN;
    }
    return(true);
   }

   // A click on decrease-simulation-speed button.
   else if(event.target == btnSlower && event.id == Event.END){
    ConfigCurrent.WAIT_TIME = ConfigCurrent.WAIT_TIME + ConfigCurrent.WAIT_TIME_PORTION;
    if (ConfigCurrent.WAIT_TIME > ConfigCurrent.WAIT_TIME_MIN){
     btnFaster.enable();
    }
    return(true);
   }

   // A click on freeze-simulation button.
   else if(event.target == btnFreeze && event.id == Event.END){
    if(bFrozen){
            simulation.unFreeze();
            bFrozen = false;
            btnFreeze.setLabel("Freeze");
     btnFaster.enable();
     btnSlower.enable();
     if (ConfigCurrent.MODE == EnumSimulationMode.INTERACTIVE){
      btnAdd.enable();
      btnEnd.enable();
     }
    }
    else{
            simulation.freeze();
            bFrozen = true;
            btnFreeze.setLabel("Un-Freeze");
     btnFaster.disable();
     btnSlower.disable();
     btnAdd.disable();
     btnEnd.disable();
    }
    return(true);
   }
   else
    return super.handleEvent(event);
  }


  /* End simulation. */

  public void endSimulation(){

   // Print out simulation results to a file, if specified.
   if (ConfigCurrent.OUTPUT_TO_FILE)
    simulation.printout();
```

```
    // Freeze simulation.
    simulation.freeze();

    // Hold on a while for freezing to take into effect.
    // Otherwise, some parts of the user interface may not be
    // re-initialized properly.
    try{
      yield.sleep(80);
    } catch (InterruptedException e) { };

    // Buttons.
    btnStart.enable();
    btnEnd.disable();
    btnAdd.disable();
    btnFaster.disable();
    btnSlower.disable();
    btnFreeze.disable();
    btnConfig.enable();
    if(bFrozen)
      btnFreeze.setLabel("Freeze");

    // Canvas.
    cvsCPU.repaint();
    cvsThroughput.repaint();
    cvsResponesT.repaint();
    cvsTurnaroundT.repaint();
    cvsWaitT.repaint();
    cvsPageFault.repaint();
    cvsFreeMemory.repaint();

    cvsCPU.setMaxVals(300);
    cvsThroughput.setMaxVals(300);
    cvsResponesT.setMaxVals(300);
    cvsTurnaroundT.setMaxVals(300);
    cvsWaitT.setMaxVals(300);
    cvsPageFault.setMaxVals(300);
    cvsFreeMemory.setMaxVals(300);

    // Lists.
    SleepList.clear();
    MsgWaitList.clear();
    SemaphoreList.clear();
    ReadyList.clear();
    HistoryList.clear();

    // Labels.
    lblRunningProc.setText("  ");
    lblCPUV.setText("  ");
    lblThroughputV.setText("  ");
    lblResponesTV.setText("  ");
    lblTurnaroundTV.setText("  ");
    lblWaitTV.setText("  ");
    lblPageFaultV.setText("  ");
    lblReadyQProcNum.setText("  ");
    lblSleepQProcNum.setText("  ");
    lblMessageQProcNum.setText("  ");
    lblSemaphoreProcNum.setText("  ");
    lblFreeMemoryV.setText("  ");
  }


  /* Starting point. */

  public static void main(String[] args){

    OSS f = new OSS();

    if (ConfigCurrent_MODE == EnumSimulationMode.INTERACTIVE)
      f.setTitle("Operating System Simulation (OSS) In Java -- Interactive");
```

```
    else
      f.setTitle("Operating System Simulation (OSS) In Java -- Non-interactive");
    bFrozen = false;

    f.resize(820, 450);

    f.show();
  }


  /* process result. */

  public void processResult(Dialog source, Object result) {

    if (source instanceof UserProcessRequestDialog) {
      UserProcessRequest userReq = (UserProcessRequest) result;
      simulation.addNewProcess(userReq);
    }

  }

  /* Add a component to the user interface according to a particular layout. */

  private void add(Component c, GridBagLayout gbl, GridBagConstraints gbc,
    int x, int y, int w, int h, int f, int a, int xw, int yw){
    gbc.gridx = x;
    gbc.gridy = y;
    gbc.gridwidth = w;
    gbc.gridheight = h;

    gbc.fill = f;
    gbc.weightx = xw;
    gbc.weighty = yw;
    gbc.anchor = a;

    gbl.setConstraints(c, gbc);
    add(c);
  }


}
```

A.24. PCB.java

```
import java.util.*;


/* This class models a process control block (PCB). */

public class PCB{

 private int id;                // Process ID.
 private int ppid;              // Parent process ID.
 private boolean free;          // Free flag.
 private int state;             // Process state.
 private byte[] registers;      // Registers.
 private int PTBR;              // Page table base register.
 private Vector frameList;      // Frame list vector.
 private UserProcess userprocess; // Simulated user process.
 private int faultingPage = 0;   // Faulting page number.
 private int quantum;           // Remaining quantum.
 private int priority;          // Execution priority.
 private int key;               // Message access key.
 private int semCounter;        // Semaphore counter.
 private boolean needSem1;      // Need-resource1 flag.
 private boolean needSem2;      // Need-resource2 flag.
 private long submitTime;       // Submitting time.
 private long firstExecutionTime; // First execution time.
 private int waitTime;          // Waiting time.
 private long enterReadyQTime;   // Entering ready queue time.
 private long exitReadyQTime;    // Exiting ready queue time.


 /* Constructor initializes PCB. */

 public PCB(int id, int ppid, boolean free, int state, int n, int PTBR,
    Vector frameList){
  this.id = id;
  this.ppid = ppid;
  this.free = free;
  this.state = state;
  this.registers = new byte[n];
  this.PTBR = PTBR;
  this.frameList = frameList;
  this.quantum = 0;
  this.priority = ConfigCurrent.PRIORITY_LEVEL;
  this.key = 0;
  this.needSem1 = false;
  this.needSem2 = false;
  this.submitTime = 0;
  this.firstExecutionTime = 0;
  this.waitTime = 0;
  this.enterReadyQTime = 0;
  this.exitReadyQTime = 0;
 }


 /* Set free flag. */

 public void freeSlot(boolean f){
  free = f;
 }


 /* Check free flag. */

 public boolean isFree(){
  return free;
 }
```

```
/* Set parent process ID. */

public void setPPID(int ppid){
 this.ppid = ppid;
}


/* Set user process. */

public void setUserPro(UserProcess userprocess){
 this.userprocess = userprocess;
}


/* Return user process. */

public UserProcess getUserPro(){
 return userprocess;
}


/* Set process state. */

public void setState(int state){
 this.state = state;
}


/* Return process state. */

public int getState(){
 return state;
}


/* Return process ID. */
public int getID(){
 return id;
}


/* Set registers. */

public void setRegisters(byte[] reg){
 System.arraycopy(reg, 0, this.registers, 0, reg.length-1);
}


/* Return registers. */

public byte[] getRegisters(){
 return registers;
}


/* Set PTBR. */

public void setPTBR(int PTBR){
 this.PTBR = PTBR;
 userprocess.setPTBR(PTBR);
}


/* Return PTBR. */

public int getPTBR(){
 return PTBR;
}
```

```
/* Set faulting page number. */

public void setFaultingPage(int faultingPage){
  this.faultingPage = faultingPage;
}


/* Return faulting page number. */

public int getFaultingPage(){
  return faultingPage;
}


/* Reset quantum. */

public void resetQuantum(){
  quantum = ConfigCurrent.QUANTUM;
}


/* Return quantum. */

public int getQuantum(){
  return quantum;
}


/* Decrease quantum. */

public void decQuantum(){
  quantum = quantum - 1;
}


/* Set execution priority. */

public void setPriority(int priority){
  this.priority = priority;
}


/* Return execution priority. */

public int getPriority(){
  return priority;
}


/* Set message access key. */

public void setKey(int key){
  this.key = key;
}


/* Return message access key. */

public int getKey(){
  return key;
}


/* Set semaphore counter. */

public void setSemCounter(int semCounter){
  this.semCounter = semCounter;
}
```

```
/* Return semaphore counter. */

public int getSemCounter(){
  return semCounter;
}


/* Return submitting time. */

public long getSubmitTime(){
  return submitTime;
}


/* Set submitting time. */

public void setSubmitTime(long submitTime){
  this.submitTime = submitTime;
}


/* Return first execution time. */

public long getFirstExecutionTime(){
  return firstExecutionTime;
}


/* Set first execution time. */

public void setFirstExecutionTime(long firstExecutionTime){
  this.firstExecutionTime = firstExecutionTime;
}


/* Return waiting time. */

public int getWaitTime(){
 return waitTime;
}


/* Update waiting time. */

public void updateWaitTime(){
  waitTime = waitTime + (int) (exitReadyQTime - enterReadyQTime);
}


/* Return entering ready queue time. */

public long getEnterReadyQTime(){
  return enterReadyQTime;
}


/* Set entering ready queue time. */

public void setEnterReadyQTime(long enterReadyQTime){
  this.enterReadyQTime = enterReadyQTime;
}


/* Return exiting ready queue time. */

public long getExitReadyQTime(){
  return exitReadyQTime;
}
```

```
  /* Set exiting ready queue time. */

  public void setExitReadyQTime(long exitReadyQTime){
    this.exitReadyQTime = exitReadyQTime;
  }


}
```

A.25. Page.java

```
/* This class models a page table entry. */

public class Page{

 private int frame;       // Memory frame number.
 private boolean valid;   // Valid flag.
 private boolean dirty;   // Dirty flag.
 private boolean reference; // Refered flag.


 /* Constructor initializes Page. */

 public Page(int frame, boolean valid, boolean dirty, boolean reference){
  this.frame = frame;
  this.valid = valid;
  this.dirty = dirty;
  this.reference = reference;
 }


 /* Set memory frame number. */

 public void setMemFrame(int frame){
  this.frame = frame;
 }


 /* Return memory frame number. */

 public int getMemFrame(){
  return frame;
 }


 /* Set valid flag. */

 public void setValid(boolean valid){
  this.valid = valid;
 }


 /* Return valid flag. */

 public boolean getValid(){
  return valid;
 }


 /* Set dirty flag. */

 public void setDirty(boolean dirty){
  this.dirty = dirty;
 }


 /* Return dirty flag. */

 public boolean getDirty(){
  return dirty;
 }


 /* Set refered flag. */

 public void setRef(boolean reference){
  this.reference = reference;
```

```
  }

  /* Return refered flag. */

  public boolean getRef(){
    return reference;
  }

}
```

A.26. PageMgr.java

```
import java.io.*;
import java.util.*;


/* This super class models a page manger. */

public abstract class PageMgr{

 protected Machine machine;          // Machine.
 protected ProcessTable processtable; // Process table.
 private Vector DMARequestQ;         // DMA request queue vector.
 protected Vector freeFrameList;     // Free frame list vector.
 protected MemoryFrame recoveredFrame; // Retrieved from free frame list.
 private int currentFrame;           // DMA target.
 protected boolean debugflag;        // Debug flag.


 /* Constructor initializes PageMgr. */

 public PageMgr(Machine machine, ProcessTable processtable){

  this.machine = machine;
  this.processtable = processtable;
  DMARequestQ = new Vector();

  freeFrameList = new Vector();
  for(int i = 0; i < ConfigCurrent.MEMORYFRAMES; i++){
   MemoryFrame freeframe = new MemoryFrame(true, i, 0, null);
   freeFrameList.addElement(freeframe);
  }
 }


 /* Make a DMA request. */

 public abstract void makeDMARequest(PCB currproc, int vpage);


 /* Check if the page is in free frame list. */

 public abstract boolean inFreeFrameList(PCB pcb, int vpage);


 /* Page replacement. */

 public abstract void pageReplacement();


 /* Free memory frames occupied by a process. */

 public abstract void freeMemoryFrames(PCB curproc);


 /* DMA interrupt handler. */

 public void DMAInterruptHandler(PCB proc){
  debug("DMAInterruptHandler");

  // Modify the page table entry for the faulting page.
  // 1. fill in memframe.
  // 2. set valid bit.
  int ptbr = proc.getPTBR();
  int fp = proc.getFaultingPage();

  Page page = machine.getPageEntry(ptbr + fp);
  page.setMemFrame(currentFrame);
```

```
  debug("FRAME# WRITTEN ONTO PAGETABLE " + currentFrame);
  page.setValid(true);
  machine.setPageEntry(ptbr + fp, page);

  // Handle next DMA request.
  if(!DMARequestQ.isEmpty()){
    DMARequest DMAreq = (DMARequest) DMARequestQ.firstElement();
    DMARequestQ.removeElementAt(0);
    currentFrame = DMAreq.getMemFrame();
    machine.DMASetup(DMAreq);
  }
}


/* Return a free memory frame. */

public MemoryFrame getFreeFrame(){
  debug("getFreeFrame");
  MemoryFrame frame = (MemoryFrame) freeFrameList.firstElement();
  freeFrameList.removeElement(frame);
  return frame;
}


/* Return the memory frame recovered from free frame list. */

public MemoryFrame getMemFrame(){
  debug("GetMemFrame");
  return recoveredFrame;
}


/* Return free frame number. */

public int getFreeFrameNumber(){
  return freeFrameList.size();
}


/* Set debug flag. */

public void setDebug (boolean flag) {
  debugflag = flag;
}


/* Initiate a DMA request. */

protected void initiateDMA(MemoryFrame frame, PCB currproc, int vpage){
  debug("initiateDMA");
  int memframe = frame.getMemFrame();
  debug("NEWLY ALLOCATED FRAME# " + memframe);
  int bsframe = currproc.getPTBR() + vpage;
  DMARequest DMAreq = new DMARequest(memframe, bsframe,
                        EnumDMATransferDirection.BSTOMEM);
  currproc.setFaultingPage(vpage);

  // One at a time.
  if(machine.isDMAPending()){
    debug("DMA PENDING.....");
    DMARequestQ.addElement(DMAreq);
  }
  else{
    debug("NO DMA PENDING.....");
    currentFrame = memframe;
    machine.DMASetup(DMAreq);
  }
}
```

```
/* Print out the free frame list.
 * Note: this is for debugging purpose. */

protected void printFreeFrameList() {

 if (debugflag){
  System.out.print("FREE FRAME LIST: ");
  Vector Fr = freeFrameList;
  MemoryFrame frame;
  for (Enumeration e = Fr.elements() ; e.hasMoreElements() ;) {
   frame = (MemoryFrame) e.nextElement();
   if(frame.getPCB() == null)
    System.out.print(frame.getMemFrame()+"(null), ");
   else
    System.out.print(frame.getMemFrame()+"(P"+frame.getPCB().getID()+
              "_"+frame. getVpage()+"), ");
  }
  System.out.println("");
 }
}


/* Clear free frame list. */

protected void clearFreeFrameList(PCB proc){

  MemoryFrame frame;
  Vector free = freeFrameList;

  if(freeFrameList.size() != 0)
   for (Enumeration e = free.elements(); e.hasMoreElements();) {
    frame = (MemoryFrame) e.nextElement();
    if(frame.getPCB() == proc){
     frame.setPCB(null);
     frame.setVpage(0);
    }
   }
}


/* Debug. */

protected void debug (String str) {
 if (debugflag)
  System.out.println( "PageMgr: " + str );
}


}
```

A.27. PageMgrFIFO.java

```
import java.io.*;
import java.util.*;


/* This class models a page manager which implements a FIFO page
 * replacement algorithm. */

public class PageMgrFIFO extends PageMgr{

 private Vector occupiedFrameList; // Occupied frame list.


 /* Constructor initializes PageMgrFIFO. */

 public PageMgrFIFO(Machine machine, ProcessTable processtable){
  super(machine, processtable);
  occupiedFrameList = new Vector();
 }


 /* Make a DMA request. */

 public void makeDMARequest(PCB currproc, int vpage){

  debug("makeDMARequest");
  debug("Before allocation .....");
  printFreeFrameList();
  printOccupiedFrameList();

  MemoryFrame freeF = getFreeFrame();     // Get a free frame.
  occupyFreeFrame(freeF, currproc, vpage); // Mark it occupied.
  initiateDMA(freeF, currproc, vpage);    // Initiate a DMA request.

  debug("after allocation .....");
  printFreeFrameList();
  printOccupiedFrameList();

  // If the number of free frames is low, recover some frames from
  // occupied frame list according to a particular page replacement
  // algorithm.
  if(freeFrameList.size() <= ConfigCurrent.FREE_FRAME_THRASHOLD) {
   debug(">>>>>>>>>>> LOW FREE FRAME!  DO SOMETHING! <<<<<<<<<<<");
   pageReplacement();
  }
 }


 /* Occupy a free frame. */

 public void occupyFreeFrame(MemoryFrame frame, PCB proc, int vpage){
  debug("setFreeFrame");
  frame.setPCB(proc);
  frame.setVpage(vpage);
  occupiedFrameList.addElement(frame);
 }


 /* Check if the page is in free frame list. */

 public boolean inFreeFrameList(PCB pcb, int vpage){
  MemoryFrame frame;
  for (Enumeration e = freeFrameList.elements() ; e.hasMoreElements() ;) {
   frame = (MemoryFrame) e.nextElement();
   if((frame.getPCB() == pcb) && (frame.getVpage() == vpage)){
    recoveredFrame = frame;
    occupiedFrameList.addElement(frame);
```

```
    if(freeFrameList.removeElement(frame)){
     if(freeFrameList.size() <= ConfigCurrent.FREE_FRAME_THRASHOLD) {
      debug(">>>>>>>>>>>> LOW FREE FRAME ! DO SOMETHING!");
      pageReplacement();
     }
    }
    debug("after recovery .....");
    printFreeFrameList();
    printOccupiedFrameList();
    return true;
   }
  }
  return false;
}

/* FIFO page replacement algorithm. */

public void pageReplacement(){

 debug("PageReplacement");

 int count = ConfigCurrent.RETRIEVED_FRAMES;
 int index;
 PCB proc;

 while (count > 0) {
  MemoryFrame frame = (MemoryFrame) occupiedFrameList.elementAt(0);
  proc = frame.getPCB();
  index = frame.getVpage() + proc.getPTBR();
  Page page = machine.getPageEntry(index);
  page.setValid(false);
  freeFrameList.addElement(frame);
  occupiedFrameList.removeElementAt(0);
  count--;
 }

 debug("after replacement .....");
 printFreeFrameList();
 printOccupiedFrameList();
}

/* Free memory frames occupied by a exiting process. */

public void freeMemoryFrames(PCB curproc){

 debug("FreeMemoryFrames");

 Page page;
 MemoryFrame frame, newframe;
 int index;
 int ptbr = curproc.getPTBR();

 clearFreeFrameList(curproc);

 // Clear occupiedFrameList.
 Vector TempFrameList = new Vector();
 for (Enumeration e = occupiedFrameList.elements(); e.hasMoreElements();) {
  frame = (MemoryFrame) e.nextElement();
  if(frame.getPCB().getID() == curproc.getID()){
   index = ptbr + frame.getVpage();
   page = machine.getPageEntry(index);
   page.setValid(false);
   newframe = new MemoryFrame(true, frame.getMemFrame(), 0, null);
   freeFrameList.addElement(newframe);
  }
  else{
   TempFrameList.addElement(frame);
  }
```

```
    }
    occupiedFrameList = TempFrameList;

    debug("after exit .....");
    printFreeFrameList();
    printOccupiedFrameList();
  }


  /* Print out occupied frame list.
   * Note: this is for debugging purpose. */

  private void printOccupiedFrameList() {
    if(debugflag){
      System.out.print("OCCUPIED FRAME LIST: ");
      Vector Fr = occupiedFrameList;
      MemoryFrame frame;
      for (Enumeration e = Fr.elements() ; e.hasMoreElements() ;) {
        frame = (MemoryFrame) e.nextElement();
        System.out.print(frame.getMemFrame()+"(P"+frame.getPCB().getID()+
                  "_"+frame.getVpage()+"), ");
      }
      System.out.println("");
    }
  }


}
```

A.28. PageMgrSC.java

```
import java.io.*;
import java.util.*;


/* This class models a page manager which implements a second-chance
 * page replacement algorithm. */

public class PageMgrSC extends PageMgr{


 /* Constructor initializes PageMgrSC. */

 public PageMgrSC(Machine machine, ProcessTable processtable){
  super(machine, processtable);
 }


 /* Make a DMA request. */

 public void makeDMARequest(PCB currproc, int vpage){
  debug("makeDMARequest");
  debug("Before allocation .....");
  printFreeFrameList();

  MemoryFrame freeF = getFreeFrame();
  initiateDMA(freeF, currproc, vpage);

  debug("after allocation .....");
  printFreeFrameList();

  if(freeFrameList.size() <= ConfigCurrent.FREE_FRAME_THRASHOLD) {
   debug(">>>>>>>>>>> LOW FREE FRAME!  DO SOMETHING! <<<<<<<<<<<");
   pageReplacement();
  }
 }


 /* Check if the page is in free frame list. */

 public boolean inFreeFrameList(PCB pcb, int vpage){
  for (Enumeration e = freeFrameList.elements() ; e.hasMoreElements() ;) {
   MemoryFrame frame = (MemoryFrame) e.nextElement();
   if((frame.getPCB() == pcb) && (frame.getVpage() == vpage)){
    recoveredFrame = frame;
    if(freeFrameList.removeElement(frame)){
     if(freeFrameList.size() <= ConfigCurrent.FREE_FRAME_THRASHOLD) {
      debug(">>>>>>>>>>> LOW FREE FRAME ! DO SOMETHING!");
      pageReplacement();
     }
     return true;
    }
   }
  }
  return false;
 }


 /* Second chance page replacement algorithm. */

 public void pageReplacement(){

  debug("PageReplacement");

  for(int i = 0; i < ConfigCurrent.PAGEENTRY; i++){
   Page page = machine.getPageEntry(i);
   if(page.getRef() == true){
```

```
      page.setRef(false);
      machine.setPageEntry(i, page);
      continue;
     }
     if((page.getRef() == false) && (page.getValid() == true)){
      int procnum = i / ConfigCurrent.NUMPAGES;
      int vpage = i % ConfigCurrent.NUMPAGES;
      int framenum = page.getMemFrame();
      PCB pcb = processtable.currPro(procnum);
      MemoryFrame frame = new MemoryFrame(true, framenum, vpage, pcb);
      freeFrameList.addElement(frame);
      page.setValid(false);
     }
    }

    debug("after replacement .....");
    printFreeFrameList();
   }

   /* Free memory frames occupied by an exiting process. */

   public void freeMemoryFrames(PCB curproc){

    debug("FreeMemoryFrames");

    int ptbr = curproc.getPTBR();
    MemoryFrame frame;

    clearFreeFrameList(curproc);

    for(int i = ptbr; i < (ptbr + ConfigCurrent.NUMPAGES); i++){
     Page page = machine.getPageEntry(i);
     if(page.getValid() == true){
      int framenum = page.getMemFrame();
      frame = new MemoryFrame(true, framenum, 0, null);
      freeFrameList.addElement(frame);
      page.setValid(false);
     }
    }
    debug("after exit .....");
    printFreeFrameList();
   }


}
```

A.29. PerformanceMonitor.java

```
import java.awt.*;

/* This class models a performance monitor which draws system
 * performance graphic. */

public class PerformanceMonitor extends Canvas {

 private Image offImage;     // Off-screen display.
 private Graphics offGraphic; // Graphics context for drawing to
                              // an off-screen image.
 private int runHistory[];   // Array of samples.
 private int iVals;          // Number of samples.
 private int iMaxVals;       // Maximum number of samples.
 private int MAX = 100;      // Maximum sample value.
 private Dimension dSelf;    // The size of this canvas.


 /* Initialize canvas. */

 public void setMaxVals(int piMaxVals){

  iMaxVals = piMaxVals;
  runHistory = new int[iMaxVals];
  iVals = 0;
  dSelf = size();
 }


 /* Set maximum sample value. */

 public void setMax(int max){
  MAX = max;
 }


 /* Add samples. */

 public void addVal(int iVal) {

  if(iVals < iMaxVals) { // Add new sample to runHistory.
   runHistory[iVals] = iVal;
   iVals++;
  }
  else { // Remove the oldest sample and add new sample to runHistory.
   int iNdx;
   for(iNdx = 0; iNdx < iMaxVals-1; iNdx++)
   runHistory[iNdx] = runHistory[iNdx+1];
   runHistory[iMaxVals -1] = iVal;
  }
 }


 /* Draw the graphic. */

 public void paint(Graphics g) {
  g.drawImage(offImage, 0, 0, null);
 }


 /* Update canvas -- draw the graphic according to samples stored in
  * the array. */

 public void  update(Graphics g)  {

  double x1;   // X coordinate of starting point of the line.
```

```
  double x2;   // X coordinate of ending point of the line.
  int y1 = 0;  // Y coordinate of starting point of the line.
  int y2;      // Y coordinate of ending point of the line.
  int iElement; // Item number.

  // Create off-screen display.
  if (offImage == null) {
   offImage = createImage(dSelf.width, dSelf.height);
   offGraphic = offImage.getGraphics();
  }

  // Initialize off-screen display.
  offGraphic.setColor(Color.black);
  offGraphic.fillRect(0,0,dSelf.width, dSelf.height);

  // Now, draw the data.

  offGraphic.setColor(Color.cyan);

  // X coordinate value advanced each time a line is drawn.
  double xSpace = (((double)dSelf.width) / ((double)(iMaxVals)));

  int height = dSelf.height;
  int width = dSelf.width;

  x1 = 0;
  x2 = 0;
  y1 = height - 1;

  for(iElement = 0; iElement < iVals; iElement++) {

   y2 = height - ((height * runHistory[iElement]) / MAX);

   if(y2 < 0)
    y2 = 0;

   if(y2 > height-1)
    y2 = height-1;

   offGraphic.drawLine((int)x1, y1, (int)x2, y2);

   x1 += xSpace;
   x2 += xSpace;
  }
  paint(g);
 }


}
```

A.30. ProcessMgr.java

```
import java.io.*;
import java.util.*;


/* This class models the process manager of an operating system.
 * It handles operations like forking a new process, dispathing
 * a new process, making a process sleep, waking up a process,
 * making a process exit. */

public class ProcessMgr{

 private Scheduler scheduler;      // Scheduler.
 private Executer executer;        // Executer.
 private PageMgr pagemgr;          // Page manager.
 private Machine machine;          // Machine.
 private Interface userInterface;  // User interface.
 private Bookkeeper bookkeeper;    // Bookkeeper.
 private History history;          // Simulation history.

 private ProcessTable processtable; // Process table.

 private Vector sleepQ;            // Sleep queue.

 private int curpid;               // Current process ID.
 private int processNum;           // Number of processes running in the
                                   // system.

 private boolean debugflag;        // Debug flag.

 private Yield yield;              // Yield execution.


 /* Constructor initializes ProcessMgr. */

 public ProcessMgr(
    ProcessTable processtable,
    Executer executer,
    PageMgr pagemgr,
    Machine machine,
    Yield yield,
    History history,
    Bookkeeper bookkeeper,
    Interface userInterface){

  this.processtable = processtable;
  this.executer = executer;
  this.pagemgr = pagemgr;
  this.machine = machine;
  curpid = ConfigCurrent.MAXPROCESS;
  sleepQ = new Vector();
  this.yield = yield;
  processNum = 0;
  this.history = history;
  this.bookkeeper = bookkeeper;
  this.userInterface = userInterface;
 }


 /* Set Scheduler. */

 public void setScheduler(Scheduler scheduler){
  this.scheduler = scheduler;
 }


 /* Fork a new process. */
```

```
public int fork(){
 debug("Fork");

 // Get a free process ID.
 int pid;
 pid = processtable.freePID();

 // Update the number of processes.
 processNum++;

 // Initialize process table entry.
 processtable.initialize(pid);

 // Increase process number by one.
 processtable.incPID();

 // Get process control block (PCB).
 PCB p =  processtable.currPro(pid);

 // For bookkeeping purpose.
 Date currDate = new Date();
 p.setSubmitTime(currDate.getTime());
 p.setFirstExecutionTime(0);

 debug("PRIORITY: "+p.getPriority());

 // Make it ready to run.
 scheduler.makeReady(p);

 return pid;
}


/* Current running process exits. */

public void exit(){
 debug("Exit");

 // Get current running process.
 PCB p =  processtable.currPro(curpid);

 // Free memory frames occupied by exiting process.
 pagemgr.freeMemoryFrames(p);

 // Set the process state to terminated.
 p.setState(EnumProcessState.Terminated);

 // Write execution history.
 history.writeHistory(p, "exits");

 // Update the number of processes running in the system.
 processNum--;

 // Bookkeeping.
 Date currDate = new Date();
 bookkeeper.addThroughput(currDate.getTime());
 bookkeeper.addTurnaroundT((int) (currDate.getTime() - p.getSubmitTime()));
 bookkeeper.addResponseT((int) (p.getFirstExecutionTime() - p.getSubmitTime()));
 bookkeeper.addWaitingT(p.getWaitTime());

 // Schedule another process to run.
 scheduler.run();
}


/* Dispatch a newly picked process. */

public void dispatch(PCB newproc){
 debug("Dispatch");
```

```
  // Bookkeeping.
  if (newproc.getID() == ConfigCurrent.MAXPROCESS)
    bookkeeper.addCPUUtilization(false);
  else
    bookkeeper.addCPUUtilization(true);

  // If no new process is picked, just return.
  PCB oldproc;
  oldproc = processtable.currPro(curpid);
  if(newproc == oldproc)
    return;

  // Set the state of newly picked process to running.
  newproc.setState(EnumProcessState.Running);

  // Update registers.
  curpid = newproc.getID();
  byte[] Registers = machine.getRegisters();
  oldproc.setRegisters(Registers);
  machine.setRegisters(newproc.getRegisters());
  oldproc.setPTBR(machine.getPTBR());
  machine.setPTBR(newproc.getPTBR());

  debug("---------- ABOUT TO DISPATCH PROCESS: " + curpid);

  history.writeHistory(newproc, "is running");

  // Method to control simulation speed.
  try{
    yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };

  // Bookkeeping.
  if ((int) newproc.getFirstExecutionTime() == 0){
    Date currDate = new Date();
    newproc.setFirstExecutionTime(currDate.getTime());
  }

  // Update running process lable on the user interface.
  userInterface.runningProcess(newproc);

  // Execute newly picked process.
  executer.executeProcess(newproc, ConfigCurrent.MAXPROCESS);
}


/* Return current running process' PCB. */

public PCB getCurrPro(){
  return processtable.currPro(curpid);
}


/* Return idel process' PCB. */

public PCB getIdlePro(){
  return processtable.currPro(ConfigCurrent.MAXPROCESS);
}


/* Current runnig process sleeps. */

public void sleep(){
  debug("sleep");

  PCB proc;

  // Get current running process.
  proc = getCurrPro();
```

```
  // Set process state to blocked.
  proc.setState(EnumProcessState.Blocked);

  // Make the DMA request.
  pagemgr.makeDMARequest(proc, machine.getFaultingPage());

  // Add to sleep queue.
  sleepQ.addElement(proc);

  // Update sleep queue on the user interface.
  userInterface.sleepQAdd(proc);

  history.writeHistory(proc, "sleeps to wait for DMA to finish");

  try{
   yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };

  int i = getCurrPro().getID();
  debug("++++++++++++++++++ PROCESS: " + i+"  IS SLEEPING");
 }


/* Wake up a process. */

 public void wakeup(){
  debug("wakeup");

  PCB proc = (PCB) sleepQ.firstElement();

  sleepQ.removeElementAt(0);

  userInterface.sleepQRemove(proc);

  history.writeHistory(proc, "wakes up from waiting for DMA to finish");

  pagemgr.DMAInterruptHandler(proc);

  scheduler.makeReady(proc);
 }


/* Check if the sleep queue is empty. */

public boolean sleepQIsEmpty() {
  if (sleepQ.size() == 0)
    return true;
  else
    return false;
 }


/* Return process table. */

public ProcessTable getProcTable(){
  return processtable;
 }


/* Return sleep queue. */

public Vector getSleepQ(){
  return sleepQ;
 }


/* Return the number of processes running in the system. */

public int getProcessNum(){
```

```
  return processNum;
 }


 /* Set debug flag. */

 public void setDebug (boolean flag) {
  debugflag = flag;
 }


 /* Debug. */

 private void debug (String str) {
  if (debugflag)
    System.out.println( "ProcessMgr: " + str );
 }


}
```

A.31. ProcessTable.java

```
import java.util.*;

/* This class manages the process table. */

public class ProcessTable{

 private PCB[] processTable;          // Process table array.
 private static int nextPID = 0;      // Process ID number.
 private UserProcessRequestQueue reqQ; // User process request queue.

 private boolean debugflag;           // Debug flag.


 /* Constructor initializes ProcessTable. */

 public ProcessTable(UserProcessRequestQueue reqQ){

  this.reqQ = reqQ;

  processTable = new PCB[ConfigCurrent.MAXPROCESS+1];

  for(int i = 0; i <= ConfigCurrent.MAXPROCESS; i++){
   Vector f = new Vector();  // Frame list.
   processTable[i] = new PCB(i,0,true,0,ConfigCurrent.NREGS,0,f);
  }

  // Initialize the idle process.
  UserProcess u = new UserProcess(new UserProcessRequest(false, false,
   false, 0), 0, ConfigCurrent.ADDRSPACESIZE, debugflag);
  processTable[ConfigCurrent.MAXPROCESS].setUserPro(u);
 }


 /* Return a free process ID. */

 public int freePID(){
  debug("freepid");

  int pid = nextPID;

  // Get next free process ID.
  while(!processTable[pid].isFree()){

   pid = (pid+1) % ConfigCurrent.MAXPROCESS;

   // Process ID is running out.
   if(pid == nextPID)
    return -1;
  }

  return pid;
 }


 /* Initialize the process table entry. */

 public void initialize(int pid){
  debug("initialize");

  int i;
  UserProcessRequest request = reqQ.getRequest();

  // Mark the PCB occupied.
  processTable[pid].freeSlot(false);
```

```
  // Make idel process the parent process.
  processTable[pid].setPPID(ConfigCurrent.MAXPROCESS);

  // PTBR value.
  i = pid * ConfigCurrent.NUMPAGES;

  // Create a new user process.
  UserProcess u = new UserProcess(request, i, ConfigCurrent.ADDRSPACESIZE, debugflag);

  // Set up the PCB.
  processTable[pid].setUserPro(u);
  processTable[pid].setPTBR(i);
  processTable[pid].resetQuantum();
  processTable[pid].setPriority(request.getPriority());
 }


 /* Increase process ID number. */

 public void incPID(){
  nextPID = (nextPID+1) % ConfigCurrent.MAXPROCESS;
 }


 /* Return the PCB indexed by process ID. */

 public PCB currPro(int pid){
  return processTable[pid];
 }


 /* Set debug flag. */

 public void setDebug (boolean flag) {
  debugflag = flag;
 }


 /* Debug. */

 private void debug (String str) {
  if (debugflag)
    System.out.println( "ProcessTable: " + str );
 }


}
```

A.32. ResourceMgr.java

```
import java.io.*;
import java.util.*;


/* Resource Management is responsible for manipulating the resources granted
 * to a process. Because of the simulated resources' logical nature, the
 * manipulation is simplified in the system. In other words, we assume that
 * resources manipulation is done after the process sleeps a while. */

public class ResourceMgr{
 private Scheduler scheduler;    // Scheduler.
 private Machine machine;        // Machine.
 private Vector freeCounters;    // Free counters for computing times
                                 // processes has manipulated resources.
 private Vector sleepQ;          // PCBs waiting for ResourceDone.
 private Yield yield;            // Yeild.
 private boolean updated;        // Updating flag for interface.
 private History history;        // History.
 private Interface userinterface; // Interface.
 private boolean debugflag;      // Debug flag.


 /* Constructor for Resource Management */

 public ResourceMgr(Scheduler scheduler, Machine machine, Yield yield,
    History history, Interface userinterface){
  this.scheduler = scheduler;
  this.machine = machine;
  this.yield = yield;
  freeCounters = new Vector();
  sleepQ = new Vector();
  Integer in;
  for (int i = 0; i < (ConfigCurrent.NUMSEM1 + ConfigCurrent.NUMSEM2); i++){
   in = new Integer(i);
   freeCounters.addElement(in);
  }
  updated = false;
  debugflag = false;
  this.history = history;
  this.userinterface = userinterface;
 }


 /* Allocate a counter for counting the time process has manipulated
  * the resources granted. Sets the status of process to Blocked.
  * Puts it into sleep queue. Calls Scheduler to pick another process to run.*/

 public void manipulateRes(PCB proc){
  debug("manipulateRes");
  debug("before allocation counter");
  if(debugflag)
   printFreeCounters();
  int counter = getFreeCounter();
  machine.setSemCounter(counter, ConfigCurrent.SEMQUANTUM);
  proc.setSemCounter(counter);
  proc.setState(EnumProcessState.Blocked);
  sleepQ.addElement(proc);
  userinterface.sleepQAdd(proc);
  history.writeHistory(proc, "waits for finishing using the resource(s)");
  updated = true;
  try{
   yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) {};

  debug("P"+proc.getID()+" get counter"+counter);
  debug("after allocation counter");
```

```
  if(debugflag){
   printFreeCounters();
   printSleepQ();
  }
  scheduler.run();
 }


 /* After using the resource(s), the process is removed from sleep queue and
  * put into ready queue. */

 public void releaseRes(){
  debug("releaseRes");
  PCB proc = (PCB) sleepQ.firstElement();
  sleepQ.removeElementAt(0);
  userinterface.sleepQRemove(proc);
  history.writeHistory(proc, "finishs using the resource(s)");
  updated = true;
  try{
   yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };

  // Put a new counter into free counter list.

  freeCounters.addElement(new Integer(proc.getSemCounter()));
  if(debugflag){
   printFreeCounters();
   printSleepQ();
  }
  proc.getUserPro().setResourceDone();
  scheduler.makeReady(proc);
 }


 /* Pick a free counter from free counter list. */

 public int getFreeCounter(){
  debug("getFreeCounter");
  Integer in = (Integer) freeCounters.elementAt(0);
  int freecounter = in.intValue();
  freeCounters.removeElementAt(0);
  return freecounter;
 }


 /*  Returns sleep queue. */

 public Vector getSleepQ(){
  return sleepQ;
 }


 /* Returns updating flag of sleep queue. */

 public boolean sleepQUpdated(){
  return updated;
 }


 /* Clears updating flag of sleep queue. */

 public void clearUpdated(){
  updated = false;
 }


 /* Assigns Debug flag. */

 public void setDebug (boolean flag) {
  debugflag = flag;
```

```
  }

  /* Prints out sleep queue. */

  private void printSleepQ(){
    PCB proc;
    System.out.print("ResourcesleepQ -> ");
    for (Enumeration e = sleepQ.elements() ; e.hasMoreElements() ;) {
      proc = (PCB) e.nextElement();
      System.out.print("P"+proc.getID()+"("+proc.getSemCounter()+")"+",");
    }
    System.out.println("");
  }

  /* Prints out free counter list. */
  private void printFreeCounters(){
    Integer in;
    System.out.print("freeCounters -> ");
    for (Enumeration e = freeCounters.elements() ; e.hasMoreElements() ;) {
      in = (Integer) e.nextElement();
      System.out.print(in.intValue()+",");
    }
    System.out.println("");
  }

  /* Prints out a information line for debugging purpose. */

  private void debug (String str) {
    if (debugflag)
      System.out.println( "ResourceMgr: " + str );
  }


}
```

A.33. ResultProcessor.java

```
import java.awt.*;

/* Result processor interface. */

interface ResultProcessor {

  public void processResult(Dialog source, Object result);

}
```

A.34. Scheduler.java

```
import java.io.*;
import java.util.*;


/* This class is responsible for picking up a process which is most
 * eligible to run. */

public abstract class Scheduler{

 protected ProcessMgr processmgr;  // Process manager.
 protected Interface userinterface; // User interface.
 protected History history;        // Execution history.
 protected Vector readyQ;          // Ready queue.

 protected Yield yield;            // Yield execution.

 protected boolean debugflag;      // Debug flag.

 /* Constructor initializes Scheduler. */

 public Scheduler(
    ProcessMgr processmgr,
    Yield yield,
    History history,
    Interface userinterface){

  this.processmgr = processmgr;
  this.yield = yield;
  this.history = history;
  this.userinterface = userinterface;

  this.readyQ = new Vector();
 }


 /* Pick up a process to run. */

 public void run(){
  debug("Run");

  // Display current running process' information.
  int i = (processmgr.getCurrPro()).getID();
  debug("++++++++++++++++++ readyQ.size:"+readyQ.size());
  if (debugflag)
    printReadyQ();
  debug("++++++++++++++++++ CURRENT RUNNING PROCESS PID:" + i);
  debug("++++++++++++++++++ QUANTUM:" + processmgr.getCurrPro().getQuantum());
  debug("++++++++++++++++++ PRIORITY:" + processmgr.getCurrPro().getPriority());
  debug("++++++++++++++++++ SleepQIsEmpty:" + processmgr.sleepQIsEmpty());

  PCB newproc = pick();       // Pick up a new process to run.
  processmgr.dispatch(newproc); // Dispatch the newly picked process.
 }


 /* Make a process ready to run. */

 public void makeReady(PCB pcb){
  debug("MakeReady");

  // Set process state ready to run.
  pcb.setState(EnumProcessState.Ready);

  // Bookkeeping.
  Date currDate = new Date();
  pcb.setEnterReadyQTime(currDate.getTime());
```

```
  // Add process to ready queue.
  readyQ.addElement(pcb);

  // Update user interface.
  userinterface.readyQAdd(pcb);

  history.writeHistory(pcb, "is ready to run");

  try{
    yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };
 }


 /* Pick up process to run. */

 public abstract PCB pick();


 /* Return ready queue. */

 public Vector getReadyQ(){
  return readyQ;
 }


 /* Set debug flag. */

 public void setDebug (boolean flag) {
  debugflag = flag;
 }


 /* Print out the ready queue.
  * Note: this is for debugging purpose. */

 protected void printReadyQ(){
  PCB proc;
  System.out.print("readyQ -> ");
  for (Enumeration e = readyQ.elements() ; e.hasMoreElements() ;) {
    proc = (PCB) e.nextElement();
    System.out.print("P"+proc.getID()+",");
  }
  System.out.println("");
 }


 /* Debug. */

 protected void debug (String str) {
  if (debugflag)
    System.out.println( "Scheduler: " + str );
 }


}
```

A.35. SchedulerPR.java

```
import java.io.*;
import java.util.*;


/* This class models a scheduler which implements priority
 * algorithm. */

public class SchedulerPR extends Scheduler{


 /* Constructor initializes SchedulerPR. */

 SchedulerPR(
    ProcessMgr processmgr,
    Yield yield,
    History history,
    Interface userinterface){

  // Call super class constructor.
  super(processmgr, yield, history, userinterface);
 }


 /* Pick up the process which is most eligible to run. */

 public PCB pick(){
  debug("Pick");

  PCB curproc, newproc;
  int curpid, curstate, curpriority;

  curproc = processmgr.getCurrPro();
  curpid = curproc.getID();
  curstate = curproc.getState();
  curpriority = curproc.getPriority();

  Date currDate;

  // The bigger the priority number, the lower the priority level.

  // The highest priority level which is initialized to the lowest
  // priority level plus one. This way, it must be replaced by
  // an actual priority level as long as the ready queue is not
  // empty.
  int maxpriority = ConfigCurrent.PRIORITY_LEVEL + 1;

  // The process which has the highest priority level.
  PCB maxpriorityproc = curproc; // No meaning, to fool compiler.

  // Pick up the process with highest priority level in the ready
  // queue.
  if(readyQ.size() != 0){
   PCB temproc;
   for (Enumeration e = readyQ.elements(); e.hasMoreElements();){
    temproc = (PCB) e.nextElement();
    if( temproc.getPriority() < maxpriority ){
            maxpriorityproc = temproc;
            maxpriority = maxpriorityproc.getPriority();
    }
   }

  if(debugflag){
   System.out.println("++++++++++++++MAX PRIORITY:" + maxpriority);
   System.out.println("++++++++++++++IT IS PROCESS:" + maxpriorityproc.getID());
  }
 }
```

```
newproc = curproc; //no meaning, to fool compiler.

// Current running process is either idle,
// blocked or terminated.
// Pick either a process with highest priority from readyQ
// or idle process if readyQ is empty.
if ((curpid == ConfigCurrent.MAXPROCESS)||
   (curstate == EnumProcessState.Terminated)||
   (curstate == EnumProcessState.MessageWait)||
   (curstate == EnumProcessState.SemaphoreWait)||
   (curstate == EnumProcessState.Blocked)) {

 if(curstate == EnumProcessState.Terminated)
  curproc.freeSlot(true);

 if(readyQ.size() == 0){

  // Pick up the idle process to run.
  newproc = processmgr.getIdlePro();
 }
 else{

  // Pick up the process with highest priority level to run.
  newproc = maxpriorityproc;

  // Bookkeeping.
  currDate = new Date();
  maxpriorityproc.setExitReadyQTime(currDate.getTime());
  maxpriorityproc.updateWaitTime();

  // Remove newly picked process from the ready queue.
  readyQ.removeElement(maxpriorityproc);

  // Update the user interface.
  userinterface.readyQRemove(maxpriorityproc);

  // Update the execution history.
  history.writeHistory(maxpriorityproc, "is picked up from ready queue");

  try{
   yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };
 }
}

// curproc is a running process (but not the idle process).
// Compare its priority with priorities of those processes
// in the readyQ to decide whether or not to preempt curproc.
if ((curstate == EnumProcessState.Running)&&
        (curpid != ConfigCurrent.MAXPROCESS)) {

 if (curpriority > maxpriority){

  if (readyQ.size() == 0){
        newproc = curproc;
  }
  else{
   newproc = maxpriorityproc;

   // Bookkeeping.
   currDate = new Date();
   maxpriorityproc.setExitReadyQTime(currDate.getTime());
   maxpriorityproc.updateWaitTime();

   // Remove newly picked process from the ready queue.
   readyQ.removeElement(maxpriorityproc);

   // Update the user interface.
   userinterface.readyQRemove(maxpriorityproc);
```

```
        // Update the execution history.
        history.writeHistory(maxpriorityproc, "is picked up from ready queue");
        try{
          yield.sleep(ConfigCurrent.WAIT_TIME);
        } catch (InterruptedException e) { };

        // Set current process state to ready to run.
        curproc.setState(EnumProcessState.Ready);

        // Bookkeeping.
        currDate = new Date();
        curproc.setEnterReadyQTime(currDate.getTime());

        // Add current process to the ready queue.
        readyQ.addElement(curproc);

        // Update the user interface.
        userinterface.readyQAdd(curproc);

        // Update the execution history.
        history.writeHistory(curproc, "is ready to run");

        try{
          yield.sleep(ConfigCurrent.WAIT_TIME);
        } catch (InterruptedException e) { };
       }
      }
      else{ // Current running process has the highest priority level.
       newproc = curproc;
      }
     }

    return newproc;
   }


}
```

A.36. SchedulerRR.java

```
import java.io.*;
import java.util.*;


/* This class models a scheduler which implements round-robin
 * algorithm. */

public class SchedulerRR extends Scheduler{


 /* Constructor initializes SchedulerRR. */

 SchedulerRR(
    ProcessMgr processmgr,
    Yield yield,
    History history,
    Interface userinterface){

  // Call super class constructor.
  super(processmgr, yield, history, userinterface);
 }


 /* Pick up the process which is most eligible to run. */

 public PCB pick(){
  debug("Pick");

  PCB curproc, newproc;
  int curpid, curstate;
  curproc = processmgr.getCurrPro();
  curpid = curproc.getID();
  curstate = curproc.getState();
  newproc = curproc;
  Date currDate;

  // Current running process is either idle,
  // blocked or terminated.
  // Pick either a process from readyQ
  // or idel process if readyQ is empty.
  if ((curpid == ConfigCurrent.MAXPROCESS)||
     (curstate == EnumProcessState.Terminated)||
     (curstate == EnumProcessState.MessageWait)||
     (curstate == EnumProcessState.SemaphoreWait)||
     (curstate == EnumProcessState.Blocked)) {

   if(curstate == EnumProcessState.Terminated)
    curproc.freeSlot(true);

   if(readyQ.size() == 0){

    // Idel process is picked.
    newproc = processmgr.getIdlePro();
   }
   else{

    // Pick the process with the longest waiting time in the
    // ready queue.
    newproc = (PCB) readyQ.firstElement();

    // Reset quantum for newly picked process.
    newproc.resetQuantum();

    // Bookkeeping.
    currDate = new Date();
    newproc.setExitReadyQTime(currDate.getTime());
```

```
    newproc.updateWaitTime();

    readyQ.removeElementAt(0);

    // Update the user interface.
    userinterface.readyQRemove(newproc);

    // Update the execution history.
    history.writeHistory(newproc, "is picked up from ready queue");

    try{
     yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException e) { };

  }
}

// curproc is a running process (but not the idle process).
// Check its quantum and readyQ to decide
// whether or not to preempt curproc.
if ((curstate == EnumProcessState.Running)&&
        (curpid != ConfigCurrent.MAXPROCESS)) {

 if (curproc.getQuantum() <= 0){

   if (readyQ.size() == 0){
    curproc.resetQuantum();
        newproc = curproc;
   }
   else{

    // Pick the process with the longest waiting time in the
    // ready queue.
    newproc = (PCB) readyQ.firstElement();

    // Reset quantum.
        newproc.resetQuantum();

    // Bookkeeping.
    currDate = new Date();
    newproc.setExitReadyQTime(currDate.getTime());
    newproc.updateWaitTime();

    readyQ.removeElementAt(0);

    // Update the user interface.
    userinterface.readyQRemove(newproc);

    // Update the execution history.
    history.writeHistory(newproc, "is picked up from ready queue");

    try{
     yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException e) { };

    // Set current process state to ready to run.
    curproc.setState(EnumProcessState.Ready);

    // Bookkeeping.
    currDate = new Date();
    curproc.setEnterReadyQTime(currDate.getTime());

    readyQ.addElement(curproc);

    // Update the user interface.
    userinterface.readyQAdd(curproc);

    // Update the execution history.
    history.writeHistory(curproc, "is ready to run");
```

```
        try{
          yield.sleep(ConfigCurrent.WAIT_TIME);
        } catch (InterruptedException e) { };

      }
     }
     else{ // Quantum of current process is not running out.
      newproc = curproc;
     }
    }

   return newproc;
  }


}
```

A.37. SemaphoreMgr.java

```
import java.io.*;
import java.util.*;


/* Semaphore management is responsible for controlling the access to
 * the shared resources. To prevent deadlock, the allocation of semaphores
 * is on a none or all basis if a process requests multiple resources
 * at the same time. */

public class SemaphoreMgr{
 private ProcessMgr processmgr;  // Process management.
 private Scheduler scheduler;    // Scheduler.
 private Machine machine;        // Machine.
 private History history;        // History.
 private Interface userinterface; // Interface.
 private Yield yield;            // Yeild.
 private int resource1;          // Resource one.
 private int resource2;          // Resource two.
 private Vector semQ;            // Semaphore waiting queue.
 private boolean updated;        // Updating flag of message waiting queue.
 private boolean debugflag;      // Debug flag.


 /* Constructor initializes Semaphore Management. */

 public SemaphoreMgr(ProcessMgr processmgr, Scheduler scheduler,
    Machine machine, Yield yield, History history, Interface userinterface){
  this.resource1 = ConfigCurrent.NUMSEM1;
  this.resource2 = ConfigCurrent.NUMSEM2;
  this.semQ = new Vector();
  this.processmgr = processmgr;
  this.scheduler = scheduler;
  this.machine = machine;
  this.yield = yield;
  this.updated = false;
  this.debugflag = false;
  this.history = history;
  this.userinterface = userinterface;
 }


 /* Semaphore Management defines two types of resources, each has several
  * instances. A process can requests either or both of these two types of
  * resources at a time. For both case, if the process' request is satisfied,
  * then the process can go further, otherwise, it will be put into
  * semaphore waiting queue. */

 public int semPop(){
  debug("SemPop");
  PCB proc = processmgr.getCurrPro();
  UserProcess userpro = proc.getUserPro();
  if (satisfied(proc)) {
   history.writeHistory(proc, "is granted the semaphores");
   if (userpro.isNeedResource1())
    resource1--;
   if (userpro.isNeedResource2())
    resource2--;
   userpro.setResourceObtained();
  }
  else
   waitResource(proc);
  if (debugflag){
   printResource1();
   printResource2();
  }
  return 0;
```

```
}

/* The process releases all the resources it has occupied. Semaphore
 * Management goes through the semaphore waiting queue to see whether
 * there are some processes being satisfied for resources request due
 * to the release. And puts those processes into ready queue. */

public int semVop(){
 debug("SemVop");
 PCB proc;
 proc = processmgr.getCurrPro();
 UserProcess userpro = proc.getUserPro();

 /* Release the resources the current running process has occupied. */

 if (userpro.isNeedResource1())
   resource1++;
 if (userpro.isNeedResource2())
   resource2++;

 if (semQ.size() != 0){
  for (Enumeration e = semQ.elements() ; e.hasMoreElements() ;) {
   proc = (PCB) e.nextElement();
   if (satisfied(proc)){
    debug("P"+proc.getID()+" is ready");
    semQ.removeElement(proc);
    userinterface.semQRemove(proc);
    history.writeHistory(proc, "gets the resource(s) needed");
    updated = true;
    try{
     yield.sleep(ConfigCurrent.WAIT_TIME);
    } catch (InterruptedException ee) { };

    scheduler.makeReady(proc);
    break;
   }
  }
 }
 return 0;
}

/* Returns semaphore waiting queue. */

public Vector getSemQ(){
 return semQ;
}

/* Returns updating flag of semaphore waiting queue. */

public boolean semQUpdated(){
 return updated;
}

/* Clears updating flag of semaphore waiting queue. */

public void clearUpdated(){
 updated = false;
}

/* Assigns debugging flag. */

public void setDebug (boolean flag) {
 debugflag = flag;
}
```

```
/* Sets status of the process to SemaphoreWait, puts it into semaphore
 * waiting queue and updates the user interface. */

private void waitResource(PCB proc){
  debug("WaitResource");
  proc.setState(EnumProcessState.SemaphoreWait);
  semQ.addElement(proc);
  userinterface.semQAdd(proc);
  history.writeHistory(proc, "waits for resource(s)");
  updated = true;
  try{
    yield.sleep(ConfigCurrent.WAIT_TIME);
  } catch (InterruptedException e) { };

  debug("P"+proc.getID()+" is in semaphore wait Q");
  if (debugflag)
    printSemQ();
}


/* Checks whether the process request about shared resources can be satisfied.
 * To prevent deadlock, the allocation of semaphores is on a none or all
 * basis if a process requests multiple resources at the same time. */

private boolean satisfied(PCB proc){
  debug("Satisfied");
  UserProcess userpro = proc.getUserPro();
  if (userpro.getResource1() <= resource1 &&
      userpro.getResource2() <= resource2)
    return true;
  else
    return false;
}


/* Prints out semaphore waiting queue. */

private void printSemQ(){
  PCB proc;
  System.out.print("Semaphore WaitQ -> ");
  for (Enumeration e = semQ.elements() ; e.hasMoreElements() ;) {
    proc = (PCB) e.nextElement();
    System.out.print("P"+proc.getID()+",");
  }
  System.out.println("");
}


/* Prints out the number of resource one. */

private void printResource1(){
  System.out.println("number of resource1: " + resource1);
}


/* Prints out the number of resource two. */

private void printResource2(){
  System.out.println("number of resource2: " + resource2);
}


/* Prints out a information line for debugging purpose. */

private void debug (String str) {
  if (debugflag)
    System.out.println( "SemaphoreMgr: " + str );
  }
}
```

A.38. Simulation.java

```
import java.util.*;
import java.awt.*;
import java.lang.*;
import java.io.*;


/* This class is responsible for setting up the OS simulation system and
 * start a simulation. */

public class Simulation extends Thread{

 private UserProcessRequestQueue reqQ;       // Request to add new process.
 private Interface           userinterface; // User interface.
 private Machine             machine;     // Machine.
 private Executer            executer;    // Executer.
 private ProcessTable        processtable; // Process table.
 private PageMgr             pagemgr;     // Page manager.
 private ProcessMgr          processmgr;  // Process manager.
 private Scheduler           scheduler;   // Scheduler.
 private MessageMgr          messagemgr;  // Message manager.
 private SemaphoreMgr        semaphoremgr; // Semaphore manager.
 private ResourceMgr         resourcemgr;  // Resource manager.
 private OS                  os;          // OS.
 private SystemCall          systemcall;  // System call.
 private History             history;     // Execution history.
 private Yield               yield;       // Yiedl execution.
 private Bookkeeper          bookkeeper;  // Bookkeeper.

 private PerformanceMonitor cvsCPU;      // CPU utilization canvas.
 private PerformanceMonitor cvsThroughput; // Throughput canvas.
 private PerformanceMonitor cvsResponesT; // Response time canvas.
 private PerformanceMonitor cvsTurnaroundT;// Turn around time canvas.
 private PerformanceMonitor cvsWaitT;    // Waiting time canvas.
 private PerformanceMonitor cvsPageFault; // Page fault rate canvas.
 private PerformanceMonitor cvsFreeMemory; // Memory availability canvas.


 /* Constructor initializes Simulation. */

 public Simulation(){

  // Set up the whole system in a specific sequence.

  reqQ = new UserProcessRequestQueue();
  yield = new Yield();
  bookkeeper = new Bookkeeper();
  history = new History(yield);
  userinterface = new Interface();

  machine = new Machine(reqQ, yield, bookkeeper);
  executer = new Executer(machine, history);
  processtable = new ProcessTable(reqQ);

  switch(ConfigCurrent.MEMORY_MANAGEMENT){
   case EnumPageReplacement.FIFO:
    pagemgr = new PageMgrFIFO(machine, processtable);
    break;
   case EnumPageReplacement.SECOND_CHANCE:
    pagemgr = new PageMgrSC(machine, processtable);
    break;
   default:
    pagemgr = new PageMgrFIFO(machine, processtable);
    break;
  }

  processmgr = new ProcessMgr(processtable, executer, pagemgr, machine,
                      yield, history, bookkeeper, userinterface);
```

```
  executer.setProcessMgr(processmgr);

  switch(ConfigCurrent.PROCESS_MANAGEMENT){
   case EnumScheduling.ROUND_ROBIN:
    scheduler = new SchedulerRR(processmgr, yield, history, userinterface);
    break;
   case EnumScheduling.PRIORITY:
    scheduler = new SchedulerPR(processmgr, yield, history, userinterface);
    break;
   default:
    scheduler = new SchedulerRR(processmgr, yield, history, userinterface);
    break;
  }

  processmgr.setScheduler(scheduler);
  messagemgr = new MessageMgr(processmgr, scheduler, machine, yield, history,
                              userinterface);
  semaphoremgr = new SemaphoreMgr(processmgr, scheduler, machine, yield,
                    history, userinterface);
  resourcemgr = new ResourceMgr(scheduler, machine, yield, history, userinterface);
  executer.setResourceMgr(resourcemgr);
  os = new OS(scheduler, processmgr, pagemgr, machine, messagemgr, semaphoremgr, resourcemgr);
  machine.setOS(os);
  systemcall = new SystemCall(machine);
  executer.setSystemCall(systemcall);
 }


/* This is the control part of simulation. */

public void run(){

 // Turn off the debugging.
 setDebug(false);

 // For non-interactive simualtion mode, processes are added into
 // the system automatically at this point of time.
 if (ConfigCurrent.MODE == EnumSimulationMode.NON_INTERACTIVE)
  for (int i=0; i<ConfigCurrent.MAXPROCESS; i++) {
   int priority = (int)(Math.random() * ConfigCurrent.PRIORITY_LEVEL);
   addNewProcess(new UserProcessRequest(true,true,true,priority));
  }

 // Start the user interface thread.
 userinterface.start();

 // Start the machine (the idel process is running).
 try{

  // Allow some time for the user interface to initialize itself.
  sleep(ConfigCurrent.WAIT_TIME);

  // Idel process is running.
  executer.idleOp();

 } catch (InterruptedException e) { };
}


/* Set up visual components. */

public void setVisualComponents(
   PerformanceMonitor cpuHistory,
   PerformanceMonitor cvsThroughput,
   PerformanceMonitor cvsResponesT,
   PerformanceMonitor cvsTurnaroundT,
   PerformanceMonitor cvsWaitT,
   PerformanceMonitor cvsPageFault,
   PerformanceMonitor cvsFreeMemory,
   java.awt.List pReadyList,
```

```
    java.awt.List pSleepList,
    java.awt.List pSemaphoreList,
    java.awt.List pMsgWaitList,
    java.awt.List pHistoryList,
    java.awt.Label lblCPUV,
    java.awt.Label lblThroughputV,
    java.awt.Label lblResponesTV,
    java.awt.Label lblTurnaroundTV,
    java.awt.Label lblWaitTV,
    java.awt.Label lblPageFaultV,
    java.awt.Label lblFreeMemoryV,
    java.awt.Label pRunProcLabel,
    java.awt.Label lblReadyQProcNum,
    java.awt.Label lblSleepQProcNum,
    java.awt.Label lblMessageQProcNum,
    java.awt.Label lblSemaphoreProcNum,
    java.awt.Button btnAdd,
    OSS oss) {

  this.cvsCPU = cpuHistory;
  this.cvsThroughput = cvsThroughput;
  this.cvsResponesT = cvsResponesT;
  this.cvsTurnaroundT = cvsTurnaroundT;
  this.cvsWaitT = cvsWaitT;
  this.cvsPageFault = cvsPageFault;
  this.cvsFreeMemory = cvsFreeMemory;

  userinterface.initialize(oss,processmgr,pagemgr,bookkeeper,reqQ,btnAdd);
  userinterface.setLabels(lblCPUV,lblThroughputV,lblResponesTV,
              lblTurnaroundTV,lblWaitTV,lblPageFaultV,lblFreeMemoryV,
              pRunProcLabel,lblReadyQProcNum,lblSleepQProcNum,
              lblMessageQProcNum,lblSemaphoreProcNum);
  userinterface.setLists(pReadyList,pSleepList,pSemaphoreList,pMsgWaitList,
                pHistoryList);
  userinterface.setCanvas(cvsCPU,cvsThroughput,cvsResponesT,cvsTurnaroundT,
                cvsWaitT,cvsPageFault,cvsFreeMemory);

  history.setDisplay(pHistoryList, true);
}


/* Freeze the execution. */

public void freeze(){
  machine.freezeExec();
  userinterface.freezeExec();
  history.freezeExec();
}


/* Un-freeze the execution. */

public void unFreeze(){
  machine.unFreeze();
  userinterface.unFreeze();
  history.unFreeze();
}


/* Add request to create new process. */

public void addNewProcess(UserProcessRequest userReq){
  reqQ.addRequest(userReq);
}


/* Print out the simulation result to a file. */

public void printout(){
  try{
```

```
    PrintStream os = new PrintStream(new FileOutputStream(ConfigCurrent.OUTPUT_FILE_NAME));
    IOFormat.print(os, "%s\n", "OPERATING SYSTEM SIMULATION (" + (new Date()).toString() + ")");
    IOFormat.print(os, "%s\n", " ");

    if (ConfigCurrent.OUTPUT_SYSTEM_INF)
      ConfigCurrent.printout(os);
    if (ConfigCurrent.OUTPUT_RESULT_INF)
      userinterface.printout(os);
    if (ConfigCurrent.OUTPUT_HISTORY_INF)
      history.printout(os);
    os.close();
   } catch(IOException e) {
      System.out.print("Error: " + e);
      System.exit(1);
    }

 }


 /* Set debug flag. */

 private void setDebug(boolean flag){
   machine.setDebug(flag);
   executer.setDebug(flag);
   processtable.setDebug(flag);
   pagemgr.setDebug(flag);
   processmgr.setDebug(flag);
   scheduler.setDebug(flag);
   os.setDebug(flag);
   systemcall.setDebug(flag);
   messagemgr.setDebug(flag);
   semaphoremgr.setDebug(flag);
   resourcemgr.setDebug(flag);
 }


}
```

A.39. SystemCall.java

```
/* This class models the system call layer of a operating system. */

public class SystemCall{

 private Machine machine;  // Machine.

 private boolean debugflag; // Debug flag.


 /* Constructor initializes SystemCall. */

 public SystemCall(Machine machine){
  this.machine = machine;
 }


 /* Fork a new process.
    Note: this function is not used in the current implementation. */

 public void fork(){
  debug("Fork");

  byte[] saveregs = new byte[ConfigCurrent.NREGS];
  byte[] registers = new byte[ConfigCurrent.NREGS];
  int retvalue;  // Return value.

  // Current registers values are backed up.
  registers = machine.getRegisters();
  System.arraycopy(registers, 0, saveregs, 0, ConfigCurrent.NREGS);

  registers[1] = (byte) EnumSystemCallNum.FORK;
  registers[2] = 0; // Program file name.
  registers[3] = 0; // Parameter 1.
  registers[4] = 0; // Parameter 2.

  machine.generateSwInterrupt();

  retvalue = registers[8];

  // Restore registers values.
  System.arraycopy(saveregs, 0, registers, 0, ConfigCurrent.NREGS);
 }


 /* Send a message. */

 public void msgSnd(int key, int sender, int receiver, String message){
  debug("Msgsnd");

  byte[] saveregs = new byte[ConfigCurrent.NREGS];
  byte[] registers = new byte[ConfigCurrent.NREGS];
  int retvalue;

  registers = machine.getRegisters();
  System.arraycopy(registers, 0, saveregs, 0, ConfigCurrent.NREGS);

  registers[1] = (byte) EnumSystemCallNum.MSGSND;
  registers[2] = (byte) key;     // Message passing key.
  registers[3] = (byte) sender;  // Sender id.
  registers[4] = (byte) receiver; // Receiver id.

  machine.generateSwInterrupt();

  retvalue = registers[8];

  System.arraycopy(saveregs, 0, registers, 0, ConfigCurrent.NREGS);
```

```
}

/* Recieve a message. */

public void msgRcv(int key){
 debug("Msgrcv");

 byte[] saveregs = new byte[ConfigCurrent.NREGS];
 byte[] registers = new byte[ConfigCurrent.NREGS];
 int retvalue;

 registers = machine.getRegisters();
 System.arraycopy(registers, 0, saveregs, 0, ConfigCurrent.NREGS);

 registers[1] = (byte) EnumSystemCallNum.MSGRCV;
 registers[2] = (byte) key; // Message passing key.

 machine.generateSwInterrupt();

 retvalue = registers[8];

 System.arraycopy(saveregs, 0, registers, 0, ConfigCurrent.NREGS);
}


/* Semaphore P operation. */

public void semPop(){
 debug("SemPop");

 byte[] saveregs = new byte[ConfigCurrent.NREGS];
 byte[] registers = new byte[ConfigCurrent.NREGS];
 int retvalue;

 registers = machine.getRegisters();
 System.arraycopy(registers, 0, saveregs, 0, ConfigCurrent.NREGS);

 registers[1] = (byte) EnumSystemCallNum.SemPop;

 machine.generateSwInterrupt();

 retvalue = registers[8];

 System.arraycopy(saveregs, 0, registers, 0, ConfigCurrent.NREGS);
}


/* Semaphore V operation. */

public void semVop(){
 debug("SemVop");

 byte[] saveregs = new byte[ConfigCurrent.NREGS];
 byte[] registers = new byte[ConfigCurrent.NREGS];
 int retvalue;

 registers = machine.getRegisters();
 System.arraycopy(registers, 0, saveregs, 0, ConfigCurrent.NREGS);

 registers[1] = (byte) EnumSystemCallNum.SemVop;

 machine.generateSwInterrupt();

 retvalue = registers[8];

 System.arraycopy(saveregs, 0, registers, 0, ConfigCurrent.NREGS);
}
```

```
  /* Set debug flag. */

  public void setDebug (boolean flag) {
    debugflag = flag;
  }


  /* Debug. */

  private void debug (String str) {
    if (debugflag)
      System.out.println( "SystemCall: " + str );
  }


}
```

A.40. UserProcess.java

```
import java.util.*;


/* This class models a user process which may have any combination of
 * memory access, message passing and semaphore operations. */

public class UserProcess{

  private int PTBR;              // PTBR.
  private Vector memoryAcc;      // Memory addresses accessed by process.
  private boolean semaphoreOp;   // Semaphore operation flag.
  private boolean messageOp;     // Message passing operation flag.
  private boolean needResource1; // Resource1 needed flag.
  private boolean needResource2; // Resource2 needed flag.
  private int resource1;         // The number of resource1 needed.
  private int resource2;         // The number of resource2 needed.
  private boolean sendMsgOp;     // Message sending operation flag.
  private boolean receiveMsgOp;  // Message receiving operation flag.
  private boolean resourceObtained; // Resources obtained flag.
  private boolean resourceDone;  // Resources manipulation done flag.
  private int msgSendKey;        // Message sending key.
  private int msgReceiveKey;     // Message receiving key.
  private Vector orderOp;        // Operation order queue.
  private boolean debugflag;     // Debug flag.


  /* Constructor initializes UserProcess. */

  public UserProcess(UserProcessRequest request, int base, int addrspac, boolean flag){

    debugflag = flag;

    PTBR = base;

    semaphoreOp = false;
    messageOp = false;

    needResource1 = false;
    needResource2 = false;

    resource1 = 0;
    resource2 = 0;

    sendMsgOp = false;
    receiveMsgOp = false;

    resourceObtained = false;
    resourceDone = false;

    msgSendKey = 0;
    msgReceiveKey = 0;

    memoryAcc = new Vector();

    // Add memory access operation to the user process.
    if(request.getMemoryOp())
      createMemoryAcc(addrspac);

    // Add semaphore operation to the user process.
    if(request.getSemaphoreOp())
      createSemaphoreOp();

    // Add message passing operation to the user process.
    if(request.getMessageOp())
      createMessageOp();
```

```
  // Randomly create the order of operations.
  orderOp = new Vector();
  createOperationOrder();

  if (debugflag)
    printOrderOp();
}


/* Check if the memory access operation is done. */

public boolean memoryAccessDone(){
  if(memoryAcc.size() == 0)
    return true;
  else
    return false;
}


/* Return the memory address visited next. */

public int memAddr(){
  Integer j = (Integer)memoryAcc.firstElement();
  int i = j.intValue();
  return i;
}


/* Remove the memory address just visited by the user process. */

public void accSucc(){
  if (memoryAcc.size() != 0)
    memoryAcc.removeElementAt(0);
}


/* Set PTBR. */

public void setPTBR(int ptbr){
  PTBR = ptbr;
}


/* Return PTBR. */

public int getPTBR(){
  return PTBR;
}


/* Check if there is a message sending operation. */

public boolean getSendMsgOp(){
  return sendMsgOp;
}


/* Clear the message sending operation flag. */

public void clearSendMsgOp(){
  sendMsgOp = false;
}


/* Return the resource obtained flag. */

public boolean getResourceObtained(){
  return resourceObtained;
}
```

```
/* Set the resource obtained flag. */

public void setResourceObtained(){
  resourceObtained = true;
}


/* Return the resource manipulation done flag. */

public boolean getResourceDone(){
  return resourceDone;
}


/* Set the resource manipulation done flag. */

public void setResourceDone(){
  resourceDone = true;
}


/* Return the message receiving operation flag. */

public boolean getReceiveMsgOp(){
  return receiveMsgOp;
}


/* Clear the message receiving operation flag. */

public void clearReceiveMsgOp(){
  receiveMsgOp = false;
}


/* Return the resource1 needed flag. */

public boolean isNeedResource1(){
  return needResource1;
}


/* Return the resource2 needed flag. */

public boolean isNeedResource2(){
  return needResource2;
}


/* Check if there is a semaphore operation. */

public boolean getSemaphoreOp(){
  return semaphoreOp;
}


/* Clear the semaphore operation flag. */

public void clearSemaphoreOp(){
  semaphoreOp = false;
}


/* Return the message sending key. */

public int getMsgSendKey(){
  return msgSendKey;
}
```

```
/* Return the message receiving key. */

public int getMsgReceiveKey(){
  return msgReceiveKey;
}


/* Remove an operation from the operation order queue. */

public void removeOp(String s){
  orderOp.removeElement(s);
}


/* Return the operation order queue. */

public Vector getOrderOp(){
  return orderOp;
}


/* Return the number of resource1. */

public int getResource1(){
  return resource1;
}


/* Return the number of resource2. */

public int getResource2(){
  return resource2;
}


/* Print out the operation order queue.
 * Note: this is for debugging purpose. */

private void printOrderOp(){
  String order;
  System.out.print("Operation order -> ");
  for (Enumeration e = orderOp.elements() ; e.hasMoreElements() ;) {
    order = (String) e.nextElement();
    System.out.print(order+",");
  }
  System.out.println("");
}


/* Create a memory access operation for the user process. */

private void createMemoryAcc(int addrspac){

  // Generate the total number of memory access.
  int total = (int)(Math.random() * ConfigCurrent.TOTAL_MEMORY_ACCESS + 1);

  debug("total pages: " + total);
  debug1("page numbers: ");

  // Generate the memory addresses.
  int i,j;
  for(i=0; i<total; i++){

    j = (int)(Math.random() * addrspac);

    debug1(j/ConfigCurrent.PAGESIZE +", ");

    Integer k = new Integer(j);
```

```
    memoryAcc.addElement(k);
   }
   System.out.println("");
  }


  /* Create a semaphore operation for the user process. */

  private void createSemaphoreOp(){

   int op = (int)(Math.random() * 3 + 1);
   semaphoreOp = true;
   switch(op){
    case 1: // Resource1 is needed.
     needResource1 = true;
     resource1 = 1;
     break;
    case 2: // Resource2 is needed.
     needResource2 = true;
     resource2 = 1;
     break;
    case 3: // Both resource1 and resource2 are needed.
     needResource1 = true;
     needResource2 = true;
     resource1 = 1;
     resource2 = 1;
     break;
   }
   debug("semaphoreOp: " + semaphoreOp);
   debug("  needResource1: " + needResource1);
   debug("  needResource2: " + needResource2);
  }


  /* Create a message passing operation for the user process. */

  private void createMessageOp(){

   messageOp = true;    // Message passing operation.

   sendMsgOp = true;    // Message sending operation.
   receiveMsgOp = true; // Message receiving operation.

   // Message sending and receiving operations have the same key.
   msgSendKey = (int)(Math.random() * ConfigCurrent.NUMKEYS);
   msgReceiveKey = msgSendKey;

   debug("messageOp: " + messageOp);
   debug("  sendmsgOp: " + sendMsgOp);
   debug("    msgsendkey: " + msgSendKey);
   debug("  receivemsgOp: " + receiveMsgOp);
   debug("    msgreceivekey: " + msgReceiveKey);
  }


  /* Add memory access operation to the operation order queue. */

  private void addMemOp(){
   orderOp.addElement((String) "Mem");
  }


  /* Add message passing operation to the operation order queue. */

  private void addMsgOp(){
   if (messageOp)
    orderOp.addElement("Msg");
  }
```

```
/* Add semaphore operation to the operation order queue. */

private void addSemOp(){
  if (semaphoreOp)
    orderOp.addElement("Sem");
}


/* Create the operation order queue. */

private void createOperationOrder(){

  int op = (int)(Math.random() * 6);
  switch(op){
    case 0:
      addMemOp();
      addMsgOp();
      addSemOp();
      break;
    case 1:
      addMemOp();
      addSemOp();
      addMsgOp();
      break;
    case 2:
      addMsgOp();
      addMemOp();
      addSemOp();
      break;
    case 3:
      addMsgOp();
      addSemOp();
      addMemOp();
      break;
    case 4:
      addSemOp();
      addMemOp();
      addMsgOp();
      break;
    case 5:
      addSemOp();
      addMsgOp();
      addMemOp();
      break;
  }
}


/* Debug. */

private void debug (String str) {
  if (debugflag)
    System.out.println( "UserProcess: " + str );
}


/* Debug. */

private void debug1 (String str) {
  if (debugflag)
    System.out.print( " " + str );
}


}
```

A.41. UserProcessRequest.java

```
/* This class contains the user supplied information which guides the system
 * to create a new user process. */

public class UserProcessRequest{

 private boolean memoryOp;   // Memory operation flag.
 private boolean messageOp;  // Message operation flag.
 private boolean semaphoreOp; // Semaphore operation flag.
 private int priority;       // Execution priority level.


 /* Constructor. */

 public UserProcessRequest (
  boolean memoryOp,
  boolean messageOp,
  boolean semaphoreOp,
  int priority) {

  this.memoryOp = memoryOp;
  this.messageOp = messageOp;
  this.semaphoreOp = semaphoreOp;
  this.priority = priority;
 }


 /* Get memory operation flag. */

 public boolean getMemoryOp() {
  return memoryOp;
 }


 /* Get message operation flag. */

 public boolean getMessageOp() {
  return messageOp;
 }


 /* Get semaphore operation flag. */

 public boolean getSemaphoreOp() {
  return semaphoreOp;
 }


 /* Get priority level. */

 public int getPriority() {
  return priority;
 }


}
```

A.42. UserProcessRequestDialog.java

```
import java.awt.*;
import java.util.*;
import java.io.*;
import java.lang.*;
import java.net.*;


/* This class is to pop up a add-new-process window. */

public class UserProcessRequestDialog extends Dialog {

 private static java.awt.Checkbox memoryOp;
 private static java.awt.Checkbox messageOp;
 private static java.awt.Checkbox semaphoreOp;
 private static java.awt.Choice priority;
 private static java.awt.Label lblPriority = new Label("Priority Level:");
 private static java.awt.Label lblOperationTypes = new Label("Operation Types:");
 private static java.awt.Label lblEmptyLine = new Label("");


 /* Constructor. */

 public UserProcessRequestDialog(OSS parent, UserProcessRequest u) {

  super(parent, "Add New User Process", true);

  // Layout management constants.
  int none = GridBagConstraints.NONE;
  int hori = GridBagConstraints.HORIZONTAL;
  int vert = GridBagConstraints.VERTICAL;
  int both = GridBagConstraints.BOTH;
  int center = GridBagConstraints.CENTER;
  int north = GridBagConstraints.NORTH;
  int northeast = GridBagConstraints.NORTHEAST;
  int east = GridBagConstraints.EAST;
  int west = GridBagConstraints.WEST;

  GridBagLayout gbl = new GridBagLayout();
  GridBagConstraints gbc = new GridBagConstraints();

  Panel p1 = new Panel();
  p1.setLayout(gbl);

  memoryOp = new Checkbox("Memory", null, u.getMemoryOp());
  messageOp = new Checkbox("Message", null, u.getMessageOp());
  semaphoreOp = new Checkbox("Semaphore", null, u.getSemaphoreOp());

  priority = new Choice();

  for (int i = 0; i <= ConfigCurrent.PRIORITY_LEVEL; i++)
   priority.addItem((new Integer(i)).toString());
  priority.select(ConfigCurrent.PRIORITY_LEVEL);

  add(p1, lblPriority, gbl, gbc, 0, 0, 1, 1, none, west, 0, 0);
  add(p1, priority, gbl, gbc, 1, 0, 1, 1, none, center, 0, 0);
  add(p1, lblEmptyLine, gbl, gbc, 0, 3, 3, 1, none, center, 0, 0);

  add(p1, lblOperationTypes, gbl, gbc, 0, 1, 1, 1, none, west, 0, 0);
  add(p1, memoryOp, gbl, gbc, 0, 2, 1, 1, none, center, 0, 0);
  add(p1, messageOp, gbl, gbc, 1, 2, 1, 1, none, center, 0, 0);
  add(p1, semaphoreOp, gbl, gbc, 2, 2, 1, 1, none, center, 0, 0);

  if (ConfigCurrent.PROCESS_MANAGEMENT == EnumScheduling.PRIORITY)
   priority.enable();
  else
   priority.disable();
```

```
 add("Center", p1);

 Panel p2 = new Panel();
 p2.add(new Button("OK"));
 p2.add(new Button("Cancel"));
 add("South", p2);
 resize(320, 150);
 move(20,10);
}


/* Handle users' interactions with the add-new-process window. */

public boolean action(Event evt, Object arg) {

 if (arg.equals("OK")) {

  dispose();

  // Process result.
  ((ResultProcessor) getParent()).processResult(this,
          new UserProcessRequest(memoryOp.getState(), messageOp.getState(),
            semaphoreOp.getState(), priority.getSelectedIndex()));

  System.out.println("priority level is: "+priority.getSelectedIndex());
 }

 else if (arg.equals("Cancel")) {
  dispose();
 }

 else
  return super.action(evt, arg);

 return true;
}


/* Event handler. */

public boolean handleEvent(Event evt) {

 if (evt.id == Event.WINDOW_DESTROY) {
  dispose();
 }

 else {
  return super.handleEvent(evt);
 }

 return true;
}


/* Use GridBagLayout to add a component. */

private void add(Panel pnl, Component c, GridBagLayout gbl,
   GridBagConstraints gbc,
   int x, int y, int w, int h, int f, int a, int xw, int yw) {
 gbc.gridx = x;
 gbc.gridy = y;
 gbc.gridwidth = w;
 gbc.gridheight = h;

 gbc.fill = f;
 gbc.weightx = xw;
 gbc.weighty = yw;
 gbc.anchor = a;
```

```
    gbl.setConstraints(c, gbc);
    pnl.add(c);
  }


}
```

A.43. UserProcessRequestQueue.java

```
import java.io.*;
import java.util.*;


/* This class models a queue which contains the user requests to create
 * new processes. */

public class UserProcessRequestQueue{

 private Vector request; // User requests queue.


 /* Constructor initializes UserProcessRequestQueue. */

 public UserProcessRequestQueue(){
  request = new Vector();
 }


 /* Check if the request queue is empty. */

 public boolean empty(){
  return request.isEmpty();
 }


 /* Return a user request with the longest waiting time. */

 public UserProcessRequest getRequest(){
  UserProcessRequest req = (UserProcessRequest) request.firstElement();
  request.removeElementAt(0);
  return req;
 }


 /* Add a user request to the request queue. */

 public void addRequest(UserProcessRequest req){
  request.addElement(req);
 }


 /* Return the size of the request queue. */

 public int size(){
  return request.size();
 }


}
```

A.44. Yield.java

```
/* This class is used by threads running in the system to
 * be nice to each others. */

public class Yield extends Thread {
}
```